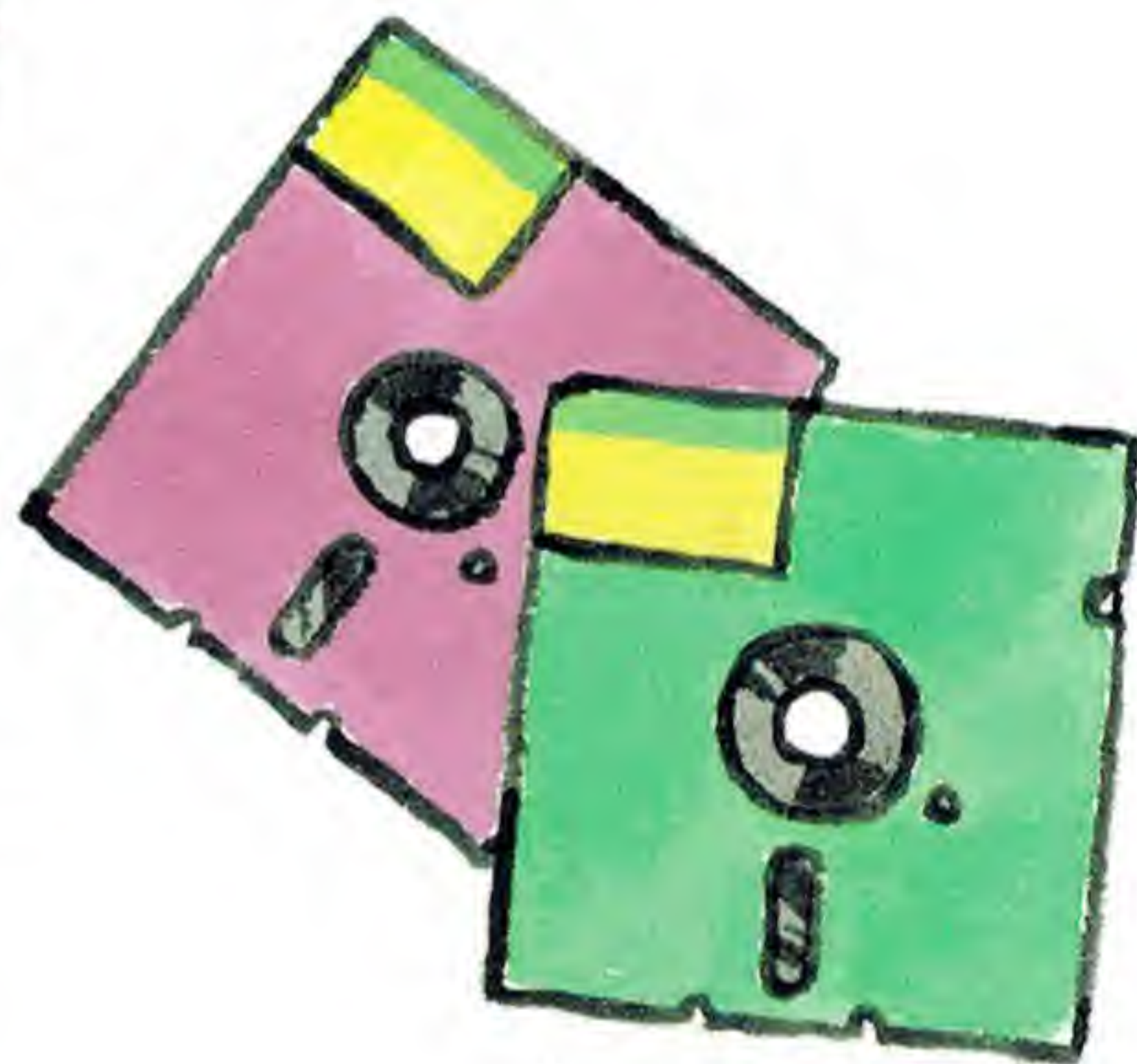


X1シリーズ  
X1/turbo/Z

# 試験に出る

ハードウェアのフルコース

祝 一平 著



一平



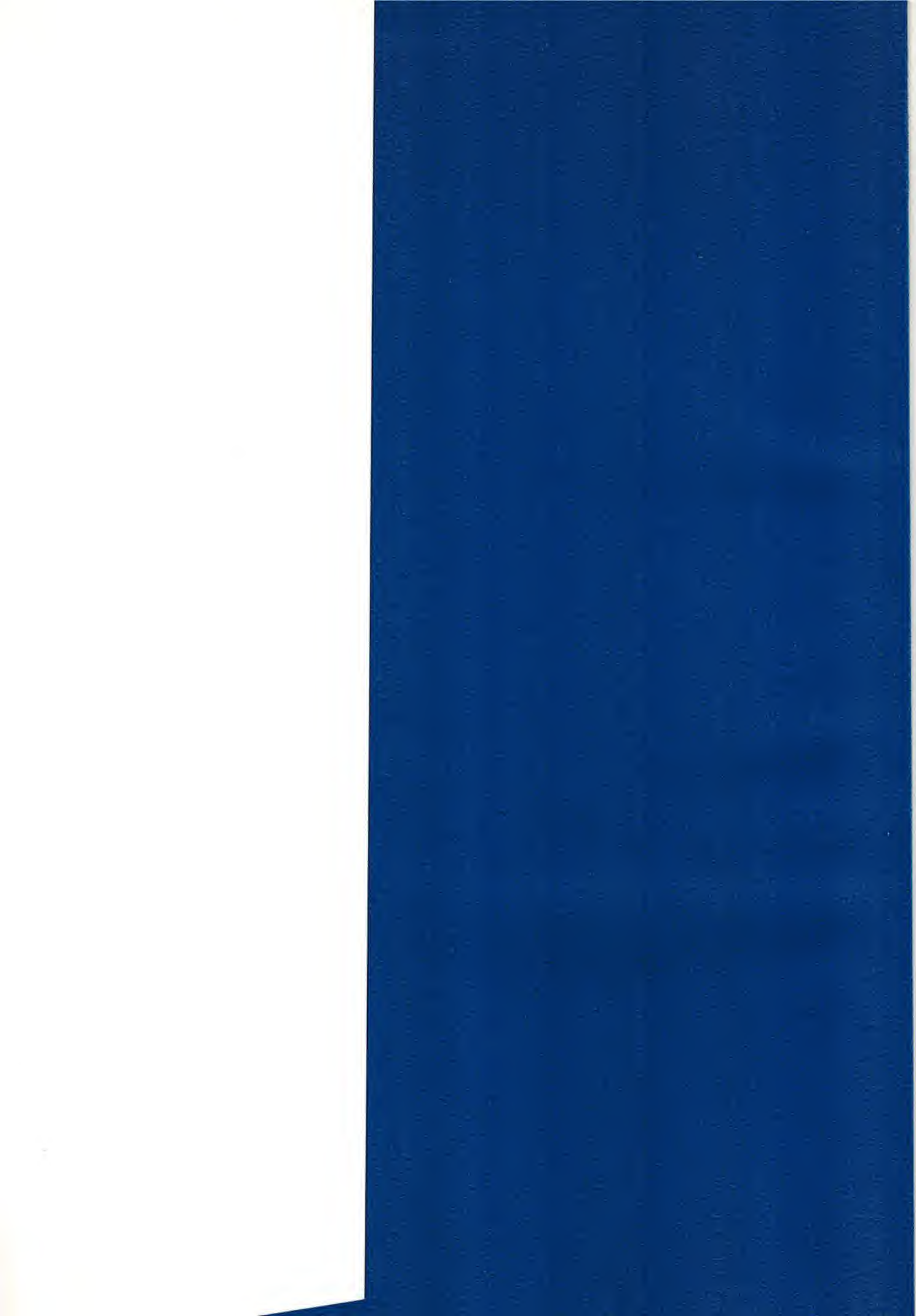
美しき哉

その筋なのは

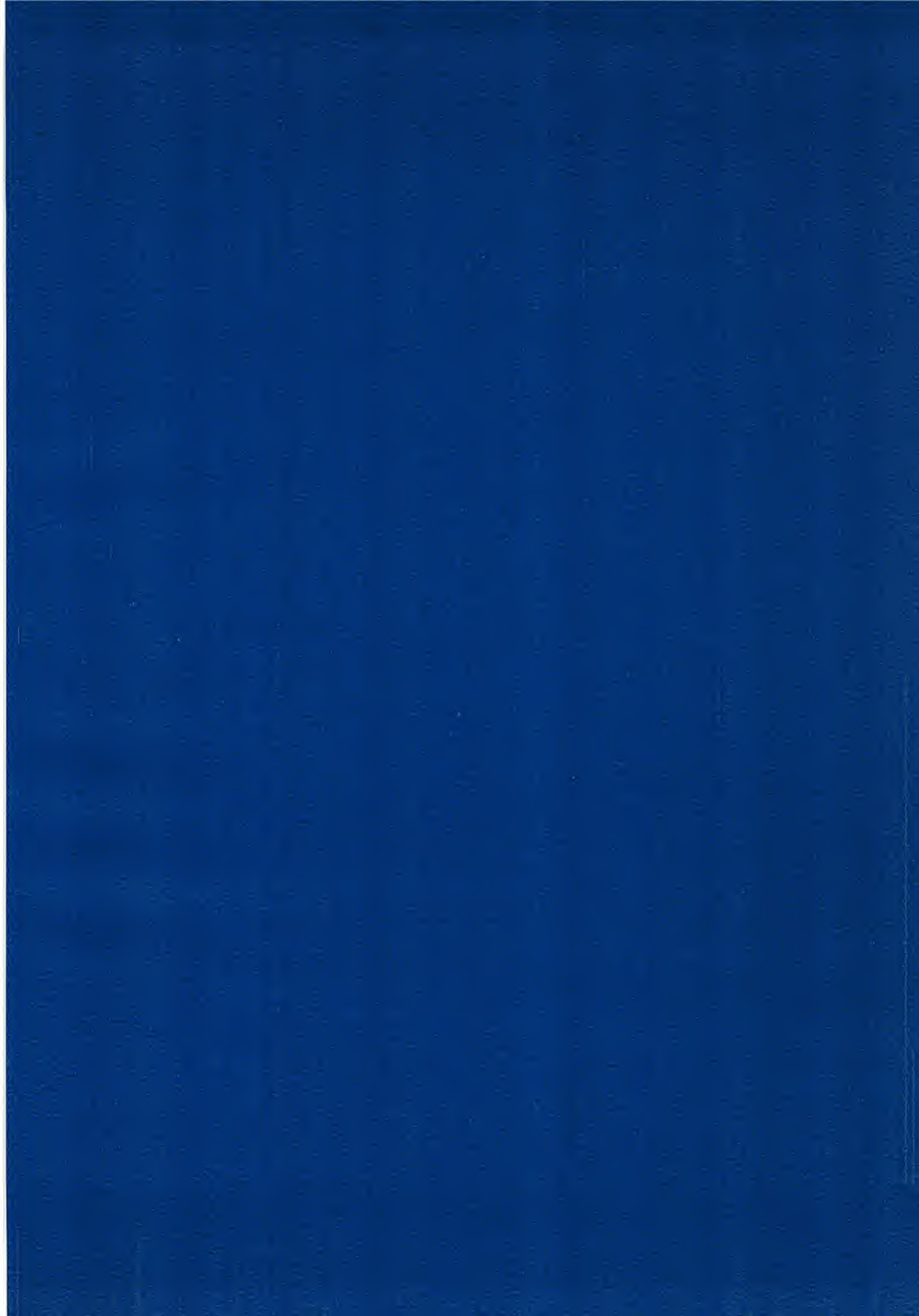


日本ソフトバンク















# 試験に出る

ハードウェアのフルコース

祝 一平 著

日本ソフトバンク



●CP/M は Digital Research 社

●HuBASIC はハドソンソフト

の登録商標です。その他プログラム名、システム名、CPU 名等は一般に各開発メーカーの登録商標です。本文中では“TM”、“®”マークは明記していません。

©1987 本書のプログラムを含むすべての内容は著作権法上の保護を受けております。著者、発行者の許諾を得ず、無断で複写、複製をすることは禁じられています。



# まえがき

'85年の6月からOh!MZ誌に連載を始め、それから2年2か月の間、X1のハードウェアをさまざまにいじくりまくった結果が本書である。ただし、連載時にあったライン描画、ペイントプログラムなどのグラフィックに関するものはページ数の関係で除外してある。世の中というものは、そんなものなのである。

連載を始めた当初は、分かっているつもりであったことも、実際に書き始めてみると意外といい加減な知識しかなかったことに気づき、あわてて調べ直したことも多々あった。それやこれやで、単純ミスからドタコまで、さまざまな間違いもあったが、そのたびに優秀な読者からご指摘をいただいた。そう考えてみると、多くのその筋なX1ユーザーのご助力を抜きにして、本書はなかったかもしれない。なんて、殊勝なことを言ってみたりする。

ともかく、本書にX1についての私の知識のほとんどを盛り込むことができた。読者各位に便利に使っていただきたい。そして、ブラックボックスではなく、すべてをユーザーに解放されたマシンの面白さを知っていただけたら幸いである。

私的ながら、本書は私にとって**1冊目**の著述にあたります。100冊を一つの頂点とするなら、ちょうどその1/100まで来たことになります。その（私にとって）記念すべき1冊にX1に巡り会えたことは、幸いなことだったと思います。

1987年 晩秋 祝 一平



# 目 次

---

第0章	I/Oマップ きっと完全無欠なI/Oマップ	9
第1章	CRTC CRTCでどすこいである ●HD46505-SPなのだ ..... 33 ●さらにCRTCを究める ..... 38 ●もっと表示するのである ..... 40 ●全画面である ..... 41	31
第2章	PCG PCGは二度おいしいのである ●PCGの基本技 ..... 44 ●PCGの定義 ..... 45 ●PCGの高速定義 ..... 50 ●CGの読み出し ..... 56 ●turboにおけるPCG/CG ..... 57	43
第3章	漢 字 漢字名野出亜留 ●漢字コードの基本 ..... 64 ●表示するのである ..... 68 ●さらに表示するのである ..... 71	63
第4章	サブCPU サブCPUのおかげなのである ●80C49なのである ..... 78 ●サブCPUの実態なのである ..... 79 ●さあて、使い方である ..... 79	77



第5章	<b>CTC</b>	97
	CTCは律儀なのである	
	● CTCの概略である	99
	● CTCなのである	99
	● 割り込みベクトル	102
	● CTCの使い方である	102
	● 実技編である	105
	● 結論である	110
第6章	<b>SIO</b>	113
	SIOでマウスである	
	● マウスである	114
	● マウスの実技である	117
第7章	<b>通 信</b>	131
	通信だってするのである	
	● それではいきなり始める	132
第8章	<b>DMA</b>	147
	DMAはヘビー級である	
	● DMAの基礎	149
	● DMAは爆発である	150
	● DMAは賢いのである	152
第9章	<b>フロッピーディスク</b>	175
	ディスクを回すのである	
	● フロッピーディスクドライブの種類	177
	● トラック、セクタなどなど	180
	● 本題に入る前に	181



# 目 次

---

●では、始める.....	181
●とりあえずFDCについて.....	187
●TYPE I の実習.....	190
●TYPE II の実習.....	196
●TYPE III.....	203
●TYPE IV.....	217
●そしてDMAである.....	217
●脱線である.....	218
●本題である.....	218
●2HDの物理フォーマットである.....	223
●グラフィックするのである.....	225
●ソフト的なフォーマットである.....	225
●さらにグラフィックするのである.....	225

<b>第10章</b>	<b>PSG</b>	<b>229</b>
	PSGは基本である	
	●サンプルなのである.....	235

<b>第11章</b>	<b>FM音源</b>	<b>241</b>
	FM音源ナハトムジーク	
	●OPMである.....	242
	●OPMの基本.....	245
	●OPMのレジスタである.....	247
	●トーンダイアラである.....	257
	●MMLである.....	260
	●MMLの機能である.....	276
	●おまけである.....	280
	●解説するのである.....	280



第12章	カラーイメージボード	283
	カラーイメージボードで取り込むのである	
	●実践するのである	285
	●気分は近未来である	289
第13章	データレコーダ	299
	テープもやってしまうのである	
	●テープ関係のI/Oアドレスである	300
	●0と1である	301
	●テープの記録フォーマットである	302
	●読んだり書いたりするのである	303
第14章	turbo Z	317
	Zの機能はおいしいのである	
	●グラフィックである	318
	●画像取り込み	324
付 録		327
	A ダンプリストチェック用プログラム	328
	B X1 処理技術者試験	330
	C 初出一覧・参考文献	335
索 引		337







第

0

章

# I/Oマップ



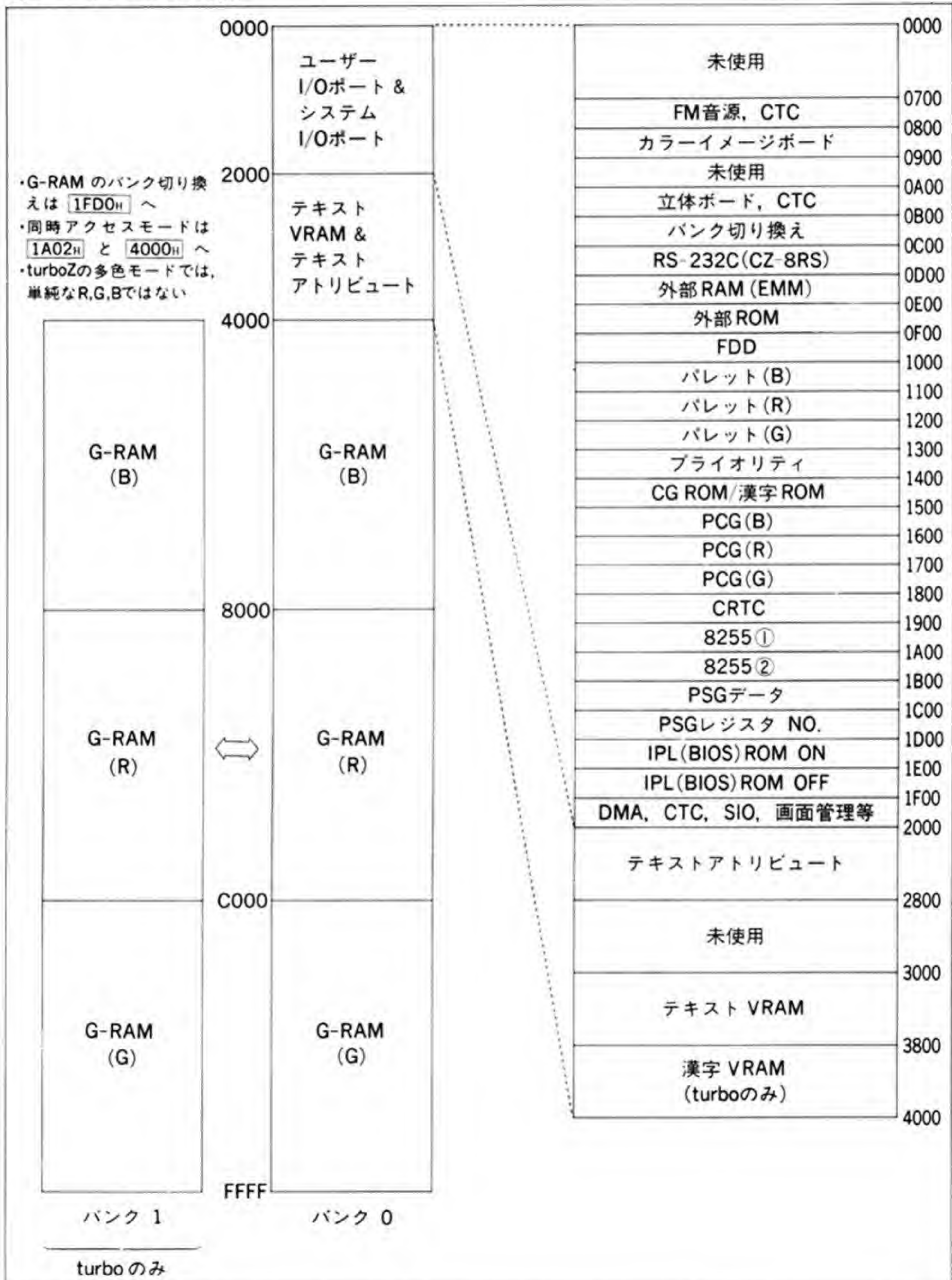
きっと完全無欠なI/Oマップ



## 第0章

# きっと完全無欠なI/Oマップ……

図 0-1 I/O マップの概観





この章では X1/X1 turbo の I/O マップを網羅するのである。この部分は Oh! MZ 誌の連載時に、初めは「たぶん完全無欠な I/O マップ」として書いたものであるが、あちこちに間違いが見つかり、やがて「ほとんど完全無欠な I/O マップ」となり、さらにバグを取って、ここに「きっと完全無欠な I/O マップ」となったのである。相変わらず完全無欠と言い切らないあたりが微妙にその筋なのである。

マップには  $\Delta V$ , **turbo**, **Z** の三つのマークがあるが、それらの意味は、

- 1)  $\Delta V$  : ノン turbo のユーザーにお勧め (もちろん turbo でも使える)
- 2) **turbo** : turbo と Z のユーザーにお勧め
- 3) **Z** : turboZ のユーザーにだけお勧め

ということである。参考にしていただきたい。そして注意しておくが、X1 の I/O ポートでは、たとえば 1802<sub>H</sub> 番地は 1800<sub>H</sub> 番地と同じである。これは業界用語で言うところの **デコード** されてないというやつである。

では、X1/X1 turbo の 64 K バイトにわたる I/O 空間を見ていくのである。

まず、図 0-1 は I/O 空間の概観である。

0000<sub>H</sub>~1FFF<sub>H</sub>までは「ユーザー I/O ポート」と「システム I/O ポート」ということになっている。しかし、困ったことにどこまでが「ユーザー」でどこまでが「システム」なのかは、よく分からないのである。一応ユーザー I/O 領域は 0000<sub>H</sub>~00FF<sub>H</sub>、システム I/O 領域は 0100<sub>H</sub>~1FFF<sub>H</sub>と考えてよいだろう。

では、番地の小さい方から順にもりもりとやってみるのである。

## 0700<sub>H</sub> FM音源/CTC (→第11章)

IN/OUT :  $\Delta V$  / **Z**

0700 <sub>H</sub>	YM2151アドレスポート	OUT
0701 <sub>H</sub>	YM2151データポート	IN/OUT
0704 <sub>H</sub>	CTCチャンネル 0	IN/OUT
0705 <sub>H</sub>	CTCチャンネル 1	IN/OUT
0706 <sub>H</sub>	CTCチャンネル 2	IN/OUT
0707 <sub>H</sub>	CTCチャンネル 3	IN/OUT

チェックポイント turboZ には FM 音源が入っているが、CTC は付いていない。ただし、0704<sub>H</sub> はデータラッチ機能があり、FM 音源機能のソフトチェックに使われる。

## 0800<sub>H</sub> カラーイメージボード (→第12章)

IN/OUT :  $\Delta V$

0800 <sub>H</sub>	カラーイメージボードコントロール	OUT
0801 <sub>H</sub>	画像データ読み込み	IN

チェックポイント Z の画像デジタイズとはまったく違う。

## 0A00<sub>H</sub> 立体ボード/CTC

IN/OUT :  $\Delta V$

0A00 <sub>H</sub>	立体ボードコントロール	OUT
0A04 <sub>H</sub>	CTCチャンネル 0	IN/OUT
0A05 <sub>H</sub>	CTCチャンネル 1	IN/OUT
0A06 <sub>H</sub>	CTCチャンネル 2	IN/OUT
0A07 <sub>H</sub>	CTCチャンネル 3	IN/OUT

## コントロール内容

コントロールデータ	D <sub>1</sub> D <sub>0</sub>	データ出力時のフィールドにおける立体スコープ動作	表示するグラフィックスクリーン
0	0 0	垂直同期ごと L, R 交互に開閉	垂直同期ごと L, R 交互に表示
1	0 1	左シャッター OPEN 右シャッター CLOSE	LEFT (PAGE0)
2	1 0	左シャッター CLOSE 右シャッター OPEN	RIGHT (PAGE1)
3	1 1	左シャッター OPEN 右シャッター OPEN	

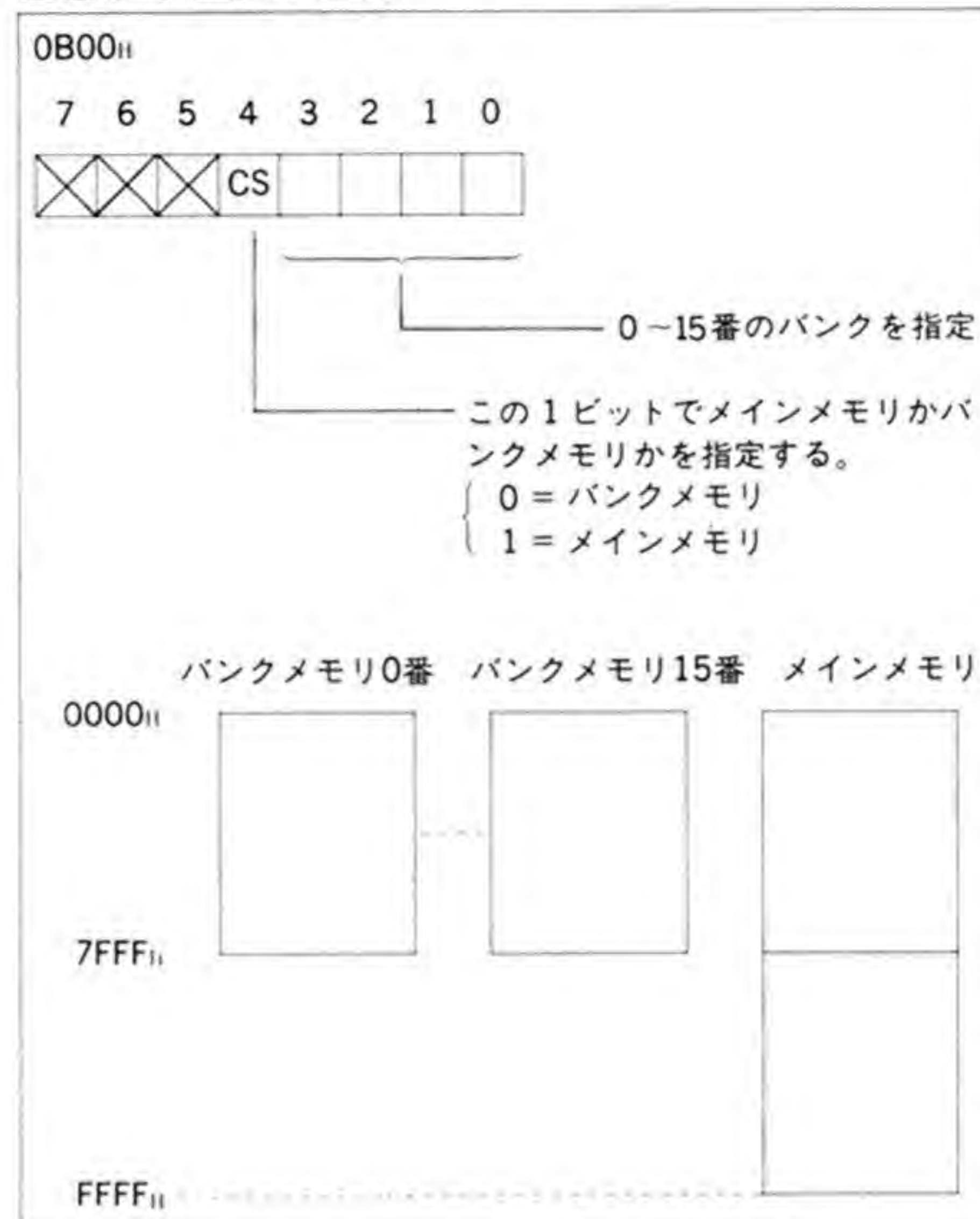
注 1) D<sub>6</sub>~D<sub>2</sub>は無効。

注 2) シャッターの OPEN/CLOSE は一度設定すると、それ以後はハードがフィールドごとの切り換えを自動的に処理してくれる。詳しくは CZ-8BR1 の取扱説明書の付録を参照のこと。



# 0B00H メイン/バンクメモリ切り換え

OUT : *turbo*



turbo で拡張された部分である。turboZ II ではバンクメモリ 0, 1 番が内蔵されている。以前は, MS(X)-DOS(Multiplan) によってバンク 0, 1 番が使われていただけであった。ノーマルな状態のバンクはシステム内メモリである。

# 0C00H RS-232Cカード(CZ-8RS)

IN/OUT :

0C*0H	データ R/W	IN/OUT
0C*1H	コントロール, ステータス R/W	IN/OUT
0C*2H	送信IEOをリセット	OUT
0C*3H	受信IEOをリセット	OUT
0C*4H	送信割り込み許可	OUT
0C*5H	送信割り込み禁止	OUT
0C*6H	受信割り込み許可	OUT
0C*7H	受信割り込み禁止	OUT

チェックポイント アドレス中の"\*"はディップスイッチで設定する。

X1 シリーズの旧タイプの RS-232C カード (CZ-8RS) である。turbo のモデル 10 以外で内蔵しているもの, および RS-232C マウスボード (CZ-8BM2) とは異なる。



## 0D00H 外部RAMボード(EMM)(→第8章)

IN/OUT: 

EMM0の場合は次のようになる。

0D00H	アドレス下位指定(00H~FFH)	OUT
0D01H	アドレス中位指定(00H~FFH)	OUT
0D02H	アドレス上位指定(00H~04H)	OUT
0D03H	データのリード/ライト(内部アドレスは自動加算)	IN/OUT

EMM1は0D04H, EMM2は0D08Hから, というように4番地ごとに計64枚が(スロットの数があれば)つながる。

## 0E00H 外部ROM(BASIC ROM, 漢字ROM)(→第3章)

IN/OUT: 

0E00H~0E03H BASIC ROM (CZ-8RB)

0E00H	アドレス上位指定	OUT
0E01H	アドレス中位指定	OUT
0E02H	アドレス下位指定	OUT
0E03H	データリード	IN

0E80H~0E82H 漢字ROM(CZ-8KR)

0E80H	左側データ/アドレス下位指定(00H~FFH)	IN/OUT
0E81H	右側データ/アドレス上位指定(00H~FFH)	IN/OUT
0E82H	(0E82H) ← 01 漢字ROMセレクト (0E82H) ← 00 増設用EPROMセレクト	OUT

0E80H~0E82H 増設用EPROM(カナ漢字変換ROM)

0E80H	ROM1データ/アドレス下位指定(00H~FFH)	IN/OUT
0E81H	ROM2データ/アドレス上位指定(00H~FFH)	IN/OUT
0E82H	(0E82H) ← 00 増設用EPROMセレクト (0E82H) ← 01 漢字ROMセレクト	OUT

BASIC ROM には, アドレスの自動カウントアップ機構はないようである。また, アドレスの指定は上位, 中位, 下位の順で外部 ROM ボードとは異なっていることに注意していただきたい。漢字 ROM のアクセス方法などに関しては第3章を見ていただきたい。増設用 EPROM は, 8K×2個がささるようになっている。HuWP(ハドソン)に付属していたカナ漢字変換ROM(I・Oデータ機器製:PIO-3055-01)である。しかしながら, X1G model 20などはそのようなROMがささる場所はなく, 「辞書はディスク上に持つ」という状況である。すなわち過去の遺物である。

## 0FD0H ハードディスク

IN/OUT: 

0FD0H	データ	IN/OUT
0FD1H	コントロール	
0FD2H		
0FD3H		



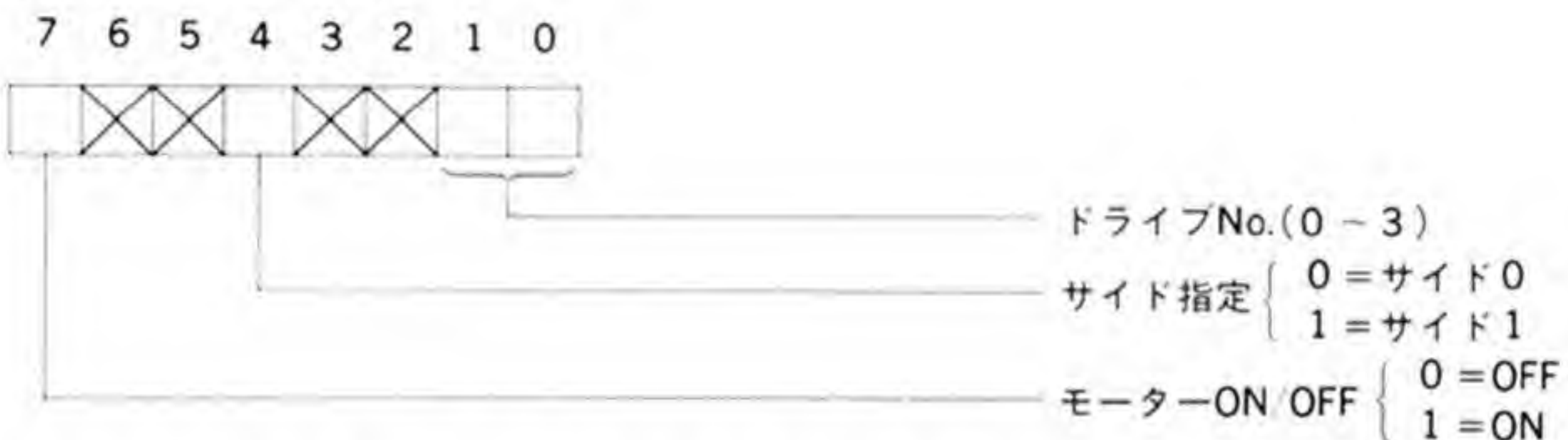
相変わらずアドレスしか分からない。ハードディスクは SCSI 準拠であるから、将来的には CD-ROM など付くはずなのだが、どーなのであろう。また、X1 にもつながるはずなのであるが、'87 年 11 月現在もサポートがない。むーん。

### 0FE8H 8インチFD

IN/OUT : *turbo*

0FE8H	ステータスレジスタ/コマンドレジスタ	IN/OUT
0FE9H	トラックレジスタ	IN/OUT
0FEAH	セクタレジスタ	IN/OUT
0FEBH	データレジスタ	IN/OUT
0FECH	FM方式指定/ドライブNo., サイド, モーターON	IN/OUT
0FEDH	MFM方式指定	IN
0FEEH	1.6Mタイプ指定	IN
0FEFH	500K・1M切り換え	IN

0FECH出力内容の意味



turbo/Z では 8 インチ FD を 4 台まで接続できるのである (5 インチ FD とは別)。しかし'87 年 11 月現在もまったくサポートされていない。5 インチ FD の 2HD が主流であるから、それも致し方ないであろう。8 インチ FD のコントロール内容は、I/O アドレスがずれている以外は 5 インチ 2HD と同じである。ただし、0FEFH 番地は無意味である。

### OFF8H 5インチFD (→第9章)

IN/OUT : *turbo*

OFF8H	ステータスレジスタ コマンドレジスタ	IN/OUT
OFF9H	トラックレジスタ	IN/OUT
OFFAH	セクタレジスタ	IN/OUT
OFFBH	データレジスタ	IN/OUT
OFFCH	FM方式指定/ドライブNo., サイド, モーターON	IN/OUT
OFFDH	MFM方式指定	IN
OFFEH	1.6Mタイプ(2HD)指定	IN
OFFFH	500K(2D)・1M(2DD)切り換え	IN

OFFCH出力内容の意味





1000<sub>H</sub>, 1100<sub>H</sub>, 1200<sub>H</sub> グラフィックパレット (→第14章)

IN/OUT:  $\Delta \nabla \nabla / \nabla$

X1/turboおよびZでのコンパチモード(OUTのみ)

この三つのI/Oポートは一組になっている。

	7	6	5	4	3	2	1	0	
10** <sub>H</sub>									(00 <sub>H</sub> ~FF <sub>H</sub> )
11** <sub>H</sub>									(00 <sub>H</sub> ~FF <sub>H</sub> )
12** <sub>H</sub>									(00 <sub>H</sub> ~FF <sub>H</sub> )

縦1列の3ビットを3桁の2進数(0~7)とみなしカラーコードにする。

たとえば、第4ビットの縦1列が

0
1
1

(右に90度回転すれば&B110=6)

は「PALET 4, 6」に対応する。

チェックポイント 標準状態(「INIT」の後)は以下のとおりである。

(10\*\*<sub>H</sub>) ← AA<sub>H</sub> = &B 1 0 1 0 1 0 1 0

(11\*\*<sub>H</sub>) ← CC<sub>H</sub> = &B 1 1 0 0 1 1 0 0

(12\*\*<sub>H</sub>) ← F0<sub>H</sub> = &B 1 1 1 1 0 0 0 0

「\*」印は何でもよいということである。またもや業界用語で言うところの「デコードされていない」である。このポートは出力のみである。うっかり IN すると、パレットの状態が変化してしまう。注意していただきたい。

turboZの多色モードの場合(IN/OUT可能)

#### 1) 4096色モード

	アドレス								データ							
	G				R				B				B'			
10	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
	G				R				B				R'			
11	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
	G				R				B				G'			
12	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

この12ビットでパレットコードを指定

この12ビット(4ビット×3回)でカラーコードを指定

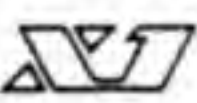
#### 2) 64色モード

上記のG, R, Bにおいて、各4ビット中の右側2ビットを00<sub>H</sub>として指定する(G, R, Bはそのまま4ビットずつ)。

チェックポイント ZでパレットR/Wには、1FC5<sub>H</sub>も参照のこと。



## 1300<sub>H</sub> プライオリティ

OUT: 

7 6 5 4 3 2 1 0

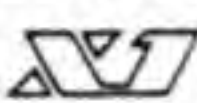
--	--	--	--	--	--	--	--

それぞれのカラーコードがテキストより優先するかを決定する。  
たとえば第5ビットが1ならば、カラーコード5(=シアン)はテキストよりも優先する。第0ビットは、背景を意味する。

BASICのPRWに指定する数値と同じである。

チェックポイント Zの場合は1FC0<sub>H</sub>も参照。

## 1400<sub>H</sub>, 1500<sub>H</sub>, 1600<sub>H</sub>, 1700<sub>H</sub>, CG, 漢字ROM, PCGアクセス (→第2,3章)

IN/OUT:  / **turbo**

X1/X1 turboコンバチアクセスモード

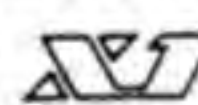
14* <sub>H</sub>	CG ROMアクセス	IN
15* <sub>H</sub>	PCG BLUEアクセス	IN/OUT
16* <sub>H</sub>	PCG RED アクセス	IN/OUT
17* <sub>H</sub>	PCG GREENアクセス	IN/OUT

X1 turbo高速アクセスモード

14*0 <sub>H</sub> ~14*F <sub>H</sub>	CG, 漢字ROMアクセス	IN
15*0 <sub>H</sub> ~15*F <sub>H</sub>	PCG BLUEアクセス	IN/OUT
16*0 <sub>H</sub> ~16*F <sub>H</sub>	PCG REDアクセス	IN/OUT
17*0 <sub>H</sub> ~17*F <sub>H</sub>	PCG GREENアクセス	IN/OUT

チェックポイント turboの場合、I/Oポート1FD0<sub>H</sub>の第5ビットが1ならば高速アクセスモード、0ならばコンバチアクセスモード。

## 1800<sub>H</sub> CRTC (→第1章)

OUT: 

1800 <sub>H</sub>	CRTCレジスタNo.指定(0~17)
1801 <sub>H</sub>	CRTCレジスタへのデータ(00 <sub>H</sub> ~FF <sub>H</sub> )

(CRTCの型番はHD46505-SP)

チェックポイント 40/80桁の切り換えには、I/Oポート1A02<sub>H</sub>の第6ビットが関係している。0なら80桁、1なら40桁。

turboでは400ラインになったことに加えて、アンダーラインモードも加わったので、1FD0<sub>H</sub> もいっしょに設定しなければならない場合がある。

18個あるCRTCのレジスタのうち実際に意味があるのは12個である。X1ではスーパーインポーズ時にスクロールできるが、これはCRTCの5番レジスタ(R5)を書き換え、さらにポート1A02<sub>H</sub>の第4ビットを0にすることで行なえる。いろいろなじくりまわすと面白い石である。



1900<sub>H</sub> サブCPU80C49(8255①)(→第4章)

IN/OUT: 

グループ	ポート端子	コントロール内容	アクティブ	
A	PA <sub>7</sub>	Z80とのデータ入出力(IN/OUT) (1900 <sub>H</sub> )	—	モード2
	PA <sub>6</sub>		—	
	PA <sub>5</sub>		—	
	PA <sub>4</sub>		—	
	PA <sub>3</sub>		—	
	PA <sub>2</sub>		—	
	PA <sub>1</sub>		—	
	PA <sub>0</sub>		—	
	PC <sub>7</sub>	Z80Aに対してデータ受け取り指示信号	L	モード0
	PC <sub>6</sub>	Z80AがポートAからデータ受け取り信号	L	
	PC <sub>5</sub>	Z80Aに対してデータ転送禁止信号	H	
	PC <sub>4</sub>	Z80AからのデータをポートAに入力/ラッチ指示信号	L	
B	PC <sub>3</sub>		—	
	PC <sub>2</sub>	カセットLEDの点灯(H:READ, L:WRITE)	—	
	PC <sub>1</sub>	Z80AへのBREAK信号	L	
	PC <sub>0</sub>	カセットのEJECTソレノイドコントロール	L	
	PB <sub>7</sub>	OBF信号	—	
	PB <sub>6</sub>	ACK信号	—	
	PB <sub>5</sub>	APSS(無記録部検出)	—	
	PB <sub>4</sub>	EJECT SWセンス	L	
	PB <sub>3</sub>		—	
	PB <sub>2</sub>	カセットテープの書き込み禁止用の爪がある状態	H	
	PB <sub>1</sub>	カセットがセットされている状態	H	
	PB <sub>0</sub>	テープエンド検出	L	

チェックポイント この図は80C49から見た場合である。このときZ80は80C49の周辺デバイスとみなされる。Z80から入出力が可能なのはポートA(PA<sub>0</sub>～PA<sub>7</sub>)だけである。

チェックポイントを繰り返すが、ポートB(PB<sub>0</sub>～PB<sub>7</sub>)、ポートC(PC<sub>0</sub>～PC<sub>7</sub>)をZ80が直接変化させることはできない。サブCPU 80C49と交信して間接的にアクセスしなければならない。結局、実際に意味があるのは1900<sub>H</sub>番地だけということであり、Z80はこのポートを使ってサブCPUと会話することになる。当然8255のモード設定などはZ80側からは行なえない。サブCPUのアクセスには8255②を参照のこと。



1A\*0H~1A\*3H 8255②

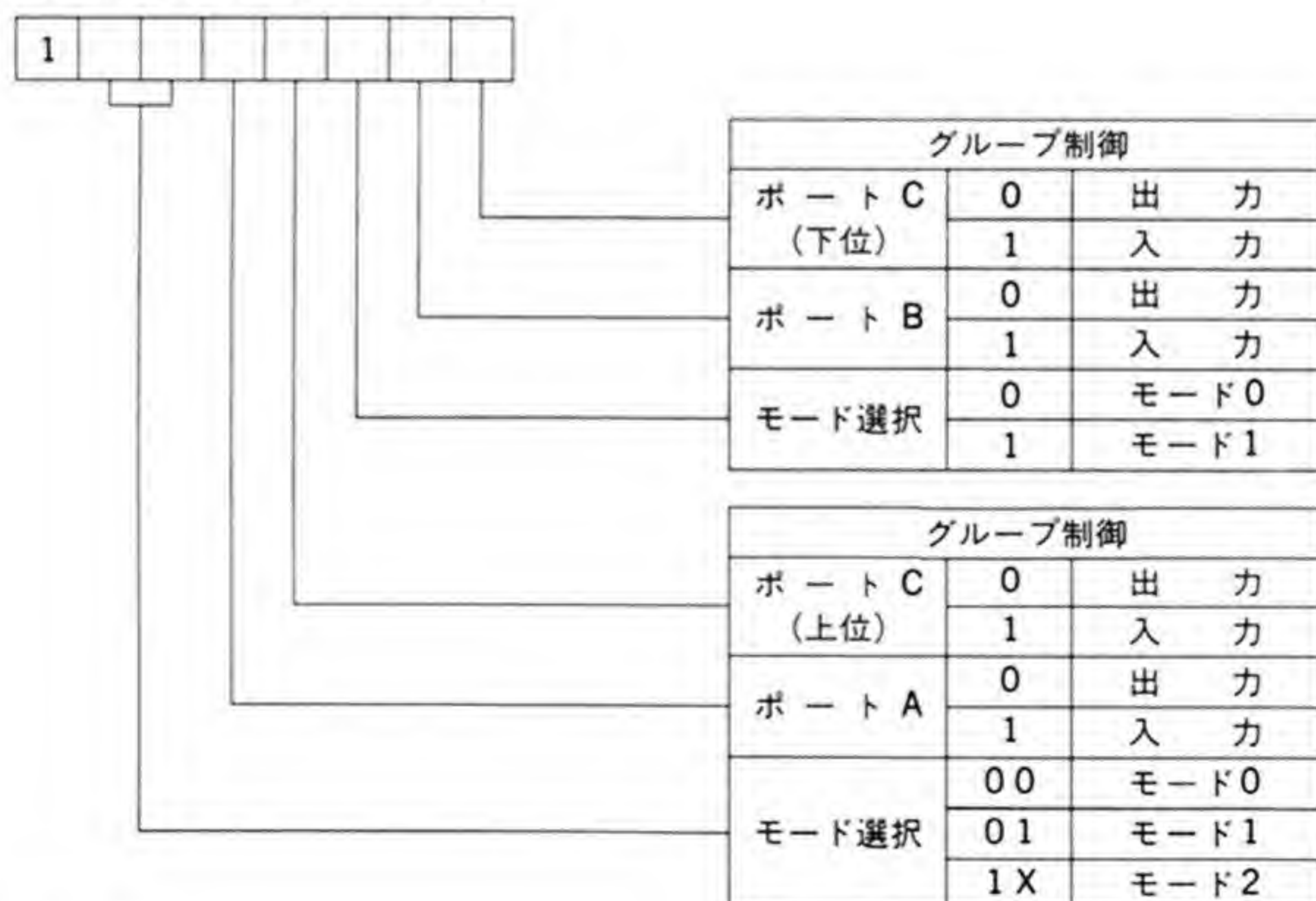
IN/OUT: 

ポート	ポート端子	コントロール内容	アクティブ	
A OUT (1A00H)	PA <sub>7</sub>	プリンタ出力データ	—	(グループA) モード0
	PA <sub>6</sub>		—	
	PA <sub>5</sub>		—	
	PA <sub>4</sub>		—	
	PA <sub>3</sub>		—	
	PA <sub>2</sub>		—	
	PA <sub>1</sub>		—	
	PA <sub>0</sub>		—	
B IN (1A01H)	PB <sub>7</sub>	垂直帰線期間信号	L	(グループB) モード0
	PB <sub>6</sub>	データ転送禁止信号	H	
	PB <sub>5</sub>	80C49からのデータ受け取り可能指示信号	L	
	PB <sub>4</sub>	BIOS ROM ON(turboのみ)	L	
	PB <sub>3</sub>	プリンタからの入力可能指示信号	L	
	PB <sub>2</sub>	垂直同期信号	H	
	PB <sub>1</sub>	カセット読み出しデータ	—	
	PB <sub>0</sub>	BREAK信号	L	
C OUT (1A02H)	PC <sub>7</sub>	立ち上がりでプリンタは入力データをサンプルする	立ち上げ	(グループA) モード0
	PC <sub>6</sub>	80/40桁 (1=40桁, 0=80桁)	—	
	PC <sub>5</sub>	立ち下げで同時アクセスモード	立ち下げ	
	PC <sub>4</sub>	スムーズスクロール信号	L	
	PC <sub>3</sub>		—	
	PC <sub>2</sub>		—	
	PC <sub>1</sub>		—	
	PC <sub>0</sub>	カセットテープへの書き込みデータ	—	
コントロールレジスタ設定ポート OUT (1A03H)				

8255モード制御(ビット7=1)

1A03H

7 6 5 4 3 2 1 0





8255ビット・セット/リセット(ビット7=0)(ポートCに対して)

1A03<sub>H</sub>

7 6 5 4 3 2 1 0



無効ビット

ビット・セット/リセット	0	リセット
	1	セット

ポートC ビット選択								
ビット	0	1	2	3	4	5	6	7
D <sub>1</sub>	0	1	0	1	0	1	0	1
D <sub>2</sub>	0	0	1	1	0	0	1	1
D <sub>3</sub>	0	0	0	0	1	1	1	1

チェックポイント ビット・セット/リセット制御はポートC(1A02<sub>H</sub>)の任意の1ビットを変化させるもので、プログラムを高速かつコンパクトにできる。

8255 ②は①と違って別に変わった使われ方はしていない。ここは重要なので少々丁寧に説明する。

プリンタへの出力は、

- 1) BUSY (PB<sub>3</sub>) = 0 まで待つ
- 2) ポート A (1A00<sub>H</sub>) へデータを出力
- 3) PC<sub>7</sub>を立ち上げる (? → 0 → 1)

で行なえる。

PB<sub>7</sub> (垂直帰線期間信号), PB<sub>2</sub> (垂直同期信号) は CRT の状態読み出し。特に PB<sub>7</sub> は PCG アクセス時に重要である。

PB<sub>6</sub> (データ転送禁止信号) はサブ CPU との会話用に使われていて、PB<sub>6</sub> = 1 はサブ CPU からの「待ってくれ」を意味する。

PB<sub>4</sub> は現在のバンクが BIOS (IPL) ROM 側か、メインメモリ側かを検出するものである。turbo のみの機能である。

PB<sub>1</sub> はカセットからの読み出しデータ。

PB<sub>0</sub> はサブ CPU からの信号で、CMT が PLAY 中に **BREAK** キーが押されたときなどに 0 になる。

PC<sub>6</sub> は 80, 40 桁の切り換え。

PC<sub>5</sub> は立ち下げ (? → 1 → 0) で同時アクセスモードになる。ただし、その前に DI で割り込みを禁止しておくこと。このモードを解除するには、何でもいから IN 命令を実行すればよい。詳しくは **4000<sub>H</sub>** へ。

PC<sub>4</sub> は 0 のとき CRTC の 5 番レジスタとともにスモースクロールを実現する (スーパーインポーズ時)。

PC<sub>0</sub> はカセットテープへの書き込みであるが、任意の長さの「1」を書けるわけではない。

なお、モード設定は IPL が行なってくれるので別にユーザーは気にする必要はない。

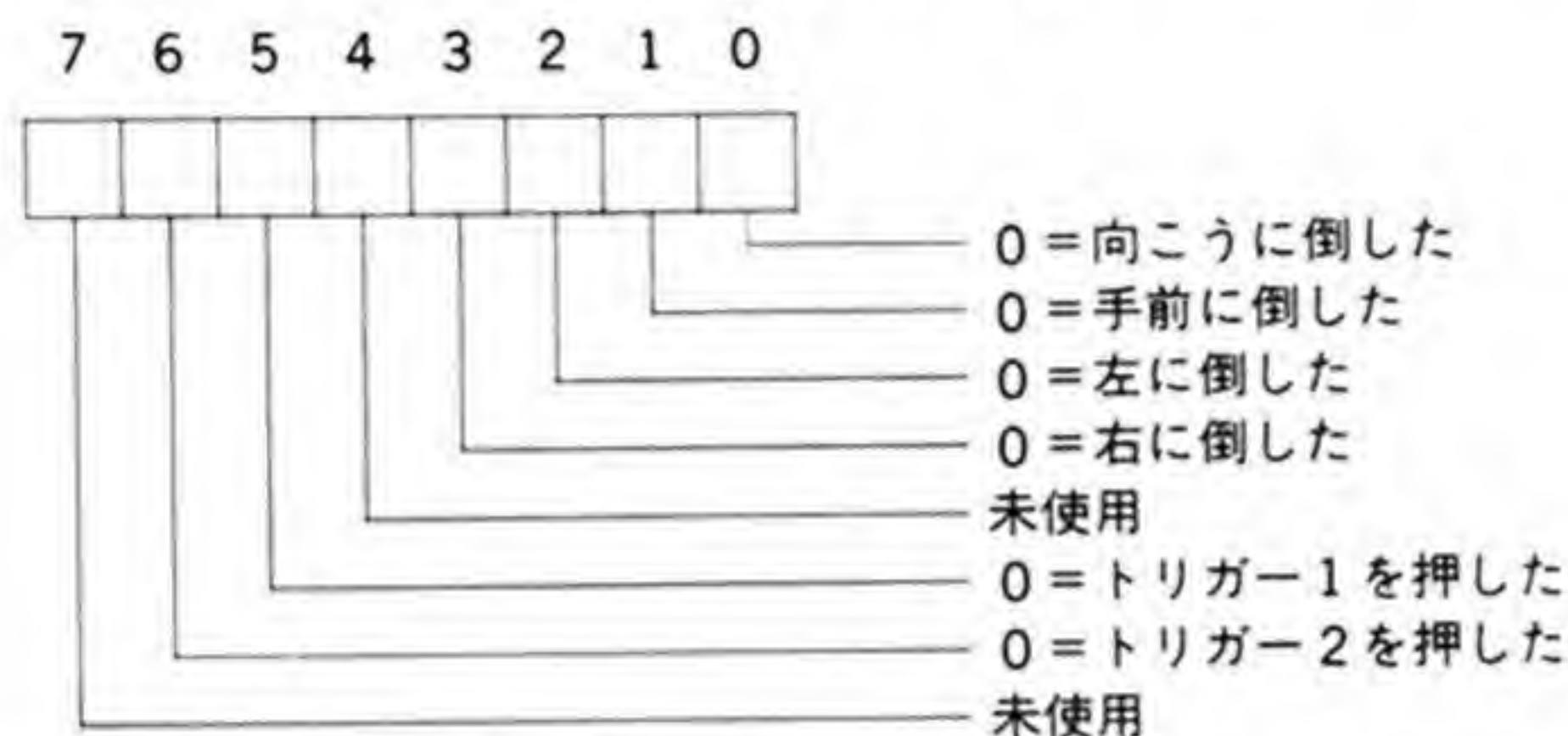


## 1B\*\*H, 1C\*\*H PSG, ジョイスティック (→第10章)

IN/OUT : 

1B**H	PSGデータ (00H~FFH)	IN/OUT
1C**H	PSGレジスタ指定 (0~15)	OUT

ジョイスティックのデータの意味 (R14, R15)



PSG (AY-3-8910) へのアクセスである。ジョイスティックをアクセスするには PSG のレジスタ, R<sub>7</sub>, R<sub>14</sub>, R<sub>15</sub>を使う。次のプログラムは, JOY1, JOY2 をともに読み出し, データを表示するものである。

```

10 OUT &H1C00, 7: 'select register
20 r5=INP(&H1B00) AND &B111111
30 OUT&H1C00, r5: 'set register
40 '
50 OUT &H1C00, 14: j1=INP(&H1B00)
60 OUT &H1C00, 15: j2=INP(&H1B00)
70 PRINT BIN$ (j1), BIN$ (j2)
80 GOTO 50

```

ジョイスティックからの入力は一方向論である。なお, レジスタ番号の指定は 1 度行なえば続けてアクセスする際には再指定する必要はない。

## 1D\*\*H, 1E\*\*H IPL (BIOS)ROM ON/OFF

OUT :  / **turbo**

1D**H	IPL ROM ON	OUT
1E**H	IPL ROM OFF	OUT

チェックポイント OUTするデータはなんでもよい。

ROMをONにすると0000H~7FFFHまでがROMに切り換わる。したがって, それを直接行なうOUT命令は8000H~の位置にななければならない。

ROMがONのとき 0 番地へジャンプすればIPLが起動する。ROMがONのときに, 0000H~7FFFHにデータの書き込みを行なうと, RAMに書き込まれる。しかし読み出しはROMから。この手法はシャドウRAMと呼ばれる。これにより64Kバイトを一発でロードできる。

1A01H番地のPB<sub>4</sub>も参照。



```
LD      BC, 1D00H
OUT     (C), A
XOR     A
LD      (100H), A
LD      A, (100H)
LD      BC, 1E00H
OUT     (C), A
```

とすると、A レジスタに何が入っているか？ 0 ではない。

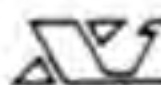
### 1F8\*<sub>H</sub> DMA (→第8章)

IN/OUT : *turbo*

1F8* <sub>H</sub>	DMAへのコマンド、データ	IN/OUT
-------------------	---------------	--------

Z80 DMA コントローラは(メモリ, I/O)↔(メモリ, I/O)間のデータ転送を高速に行なうための LSI である。DMA にはデータ転送ばかりではなくサーチ機能もある。また転送でも特定の番地の内容を、ある範囲にコピーすることができる。すなわち G-RAM 全域に 0 を転送することも可能である。この手法を使えば高速画面クリア、スクロールを実現できるが、turbo BASIC では使用していないようである。

### 1F90<sub>H</sub> SIO (→第6,7章)

IN/OUT :  / *turbo*

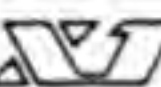
1F90 <sub>H</sub>	チャンネルAデータ	IN/OUT
1F91 <sub>H</sub>	チャンネルA制御	IN/OUT
1F92 <sub>H</sub>	チャンネルBデータ	IN/OUT
1F93 <sub>H</sub>	チャンネルB制御	IN/OUT

チェックポイント チャンネルBはマウスにつながっている。ボーレートは4800bpsである。

RS-232C カード, CZ-8RS とは互換性がないことに注意が必要。

CZ-8BM2 上の SIO のアドレスは 1F98<sub>H</sub>~1F9B<sub>H</sub>が割り当てられている。このカードには CTC が入っていて、こちらの方のアドレスは 1FA8<sub>H</sub>~1FAB<sub>H</sub>となる。これは CTC がボーレートジェネレータの役割をしているためである。

### 1FA0<sub>H</sub> CTC (→第5章)

IN/OUT :  / *turbo*

1FA0 <sub>H</sub>	チャンネル0	タイマモード	IN/OUT
1FA1 <sub>H</sub>	チャンネル1	SIOチャンネルAクロック	IN/OUT
1FA2 <sub>H</sub>	チャンネル2	SIOチャンネルB(マウス)クロック	IN/OUT
1FA3 <sub>H</sub>	チャンネル3	カウンタモード	IN/OUT

チェックポイント チャンネル0の使うクロックは4MHz, タイマ周期は4μsec~16.384msecまで。チャンネル1, 2は2MHzのクロックを使用。チャンネル3はチャンネル0をカウントし、最長タイマは4.194sec。



SIO の所で説明したように、1F98<sub>H</sub>～1F9B<sub>H</sub>にもう一つの CTC を付けることができる。また、FM 音源ボード、立体ボードなどにも CTC が載る。

### 1FB0<sub>H</sub> Zモード指定 (→第14章)

IN/OUT : Z

データ内容	コントロール
ビット 0	0 = インタレーススーパーインポーズしない 1 = インタレーススーパーインポーズする
ビット 1	無効
ビット 2	0 = 画像取り込みの階調ノーマル 1 = 画像取り込みの階調反転 } (ビット 7, 3 = 1 のときのみ有効)
ビット 3	0 = 画像取り込みをしない 1 = 画像取り込みをする } (ビット 7 = 1 のときのみ有効)
ビット 4	0 = 4096色1画面モード指定 1 = 64色2画面モード指定 } (320×200のときのみ有効)
ビット 5	無効
ビット 6	無効
ビット 7	0 = X1/X1 turboコンバチモード 1 = 多色(turboZ)モード

### 1FB9<sub>H</sub>～1FBF<sub>H</sub> Zテキストパレット指定 (→第14章)

IN/OUT : Z

アドレス	コントロール
1FB9 <sub>H</sub>	青のカラーコード
1FBA <sub>H</sub>	赤のカラーコード
1FBB <sub>H</sub>	マゼンタのカラーコード
1FBC <sub>H</sub>	緑のカラーコード
1FBD <sub>H</sub>	シアンのカラーコード
1FBE <sub>H</sub>	黄のカラーコード
1FBF <sub>H</sub>	白のカラーコード

設定データ



RGB各2ビットずつ=64色を指定できる。

### 1FC0<sub>H</sub> Zプライオリティ指定 (→第14章)

IN/OUT : Z

データ内容	コントロール
ビット 0, 1	0, 0 = テキストはグラフィックより優先 0, 1 = グラフィックはテキストより優先 1, 0 = テキストはグラフィック2面の間に入る 1, 1 = 未定義



ビット2	無効
ビット3	0=バンク0はバンク1より優先 1=バンク1はバンク0より優先
ビット4	0=片方のバンクだけを表示する 1=バンク0, 1を同時に表示する
ビット5 ビット6 ビット7	無効

ビット3, 4は2画面モード(→1FB0<sub>H</sub>番地のビット4)のときのみ有効。

- ・多色(turboZ)かつ320×200モードでのみ意味のあるポート
- ・ビット4=0のときはビット1=0とみなされる
- ・ビット4=1のときは1FD0<sub>H</sub>のビット3は無効

### 1FC1<sub>H</sub> Z画像取り込み位置補正指定 (→第14章)

IN/OUT: Z

7 6 5 4 3 2 1 0

--	--	--	--	--	--	--	--

0-255の補正ドット数を指定する。

- ・200ラインモード(1FD0<sub>H</sub>のビット0=0)のときのみ有効

### 1FC2<sub>H</sub> Zモザイク/量子化取り込み指定 (→第14章)

IN/OUT: Z

7 6 5 4 3 2 1 0

--	--	--	--	--	--	--	--

X方向モザイク

ビット2	ビット1	ビット0	ドット数
0	0	0	1
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	—

Y方向モザイク

ビット5	ビット4	ビット3	ドット数
0	0	0	1
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	—
1	1	1	—

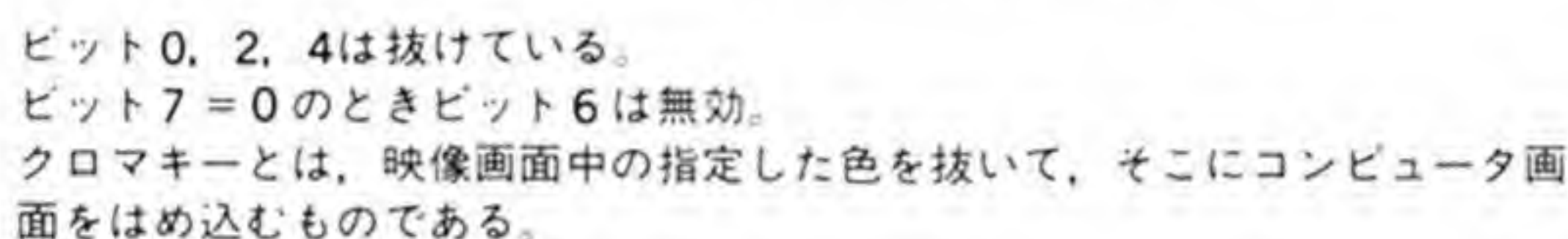
量子化指定

ビット7	ビット6	階調
0	0	4ビット階調取り込み(4096色)
0	1	3ビット階調取り込み(512色)
1	0	2ビット階調取り込み(64色)
1	1	1ビット階調取り込み(8色)

チェックポイント 64色モードを指定してある場合はビット7=1として扱われる。

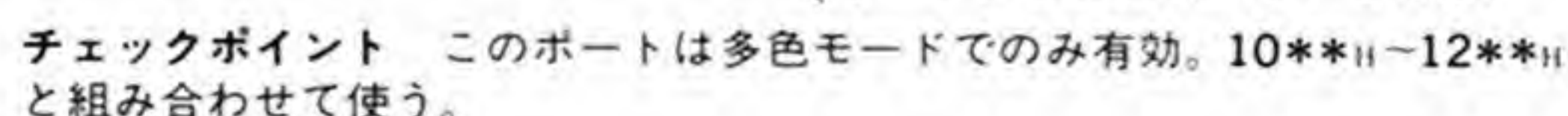


IN/OUT :  $\mathbb{Z}$

IN/OUT :  $\mathbb{Z}$ 

**チェックポイント** ビット 3 = 0 のときビット 0 ~ 2 は無効。これは、スーパーインポーズと組み合わせて、「一度だけスクロール」を実現するものである。CRTC と 8255② も必要。

IN/OUT :  $\mathbb{Z}$





## 1FD0H 画面管理 (→第1章)

(IN)/OUT : **turbo**

データ内容	コントロール
ビット 0	0 = 低解像度モニタ (200ライン) 1 = 高解像度モニタ (400ライン) モニタ切り換え
ビット 1	0 = 1本ラスタ/ドット 1 = 2本ラスタ/ドット
ビット 2	0 = ノーマル (8ラスタ/CHAR) (25行, 20行) 1 = 漢字 (16ラスタ/CHAR) (12行, 10行)
ビット 3	0 = バンク 0 表示 1 = バンク 1 表示
ビット 4	0 = バンク 0 アクセス 1 = バンク 1 アクセス
ビット 5	0 = PCG コンパチアクセス 1 = PCG 高速アクセス
ビット 6	0 = 8ラスタCGアクセス 1 = 16ラスタCGアクセス
ビット 7	0 = アンダーラインなし 1 = アンダーラインあり

**チェックポイント** BASIC起動直後はビット 1 = 1 である。これによりグラフィックを高解像度モニタで200ラインとして扱える(文字の方は400ライン)。turboでは、グラフィックが400ラインモードのときは偶数段目がバンク 0、奇数段目がバンク 1 の内容が表示されている。ビット 1 が 1 であるということは、そのとき表示されているバンク(ビット 3 で指定)の内容のパターンを下の段にも繰り返し表示させるものである。

おそらくこの説明では理解できないだろうから、次のプログラムをサンプルとして示す。

```

10 OPTION SCREEN 0 : WIDTH 80,25,0,2
20 GRAPH 0,0,0 : CLS 4 : SYMBOL(100,30), "バンク0", 5,5,1,0,PSET
30 GRAPH 2,2,0 : CLS 4 : SYMBOL(100,70), "バンク1", 5,5,2,0,PSET
40 OUT &H1FD0,&B1100001 : A$ = INKEY$ (1)
50 OUT &H1FD0,&B1100011 : A$ = INKEY$ (1)
60 OUT &H1FD0,&B1101001 : A$ = INKEY$ (1)
70 OUT &H1FD0,&B1101011 : A$ = INKEY$ (1)

```

また、turboではOUTだけだが、ZではINも可能。

ここは結構複雑である。ビット 1 はサンプルプログラムで理解して欲しい。ビット 2 はテキストの 25 (or 20) → 12 (or 10) 行の指定に使う。CRTC の設定と一緒に操作しなければ表示が乱れる。ビット 3, 4 は簡単だと思われるが念のために言うと、640×400 のモードのときビット 3 はどうでもよい。ビット 1 が 0 なら 400 ラインを表示する。ビット 5 は PCG のアクセス方式を変えるということで、PCG にアクセスするには設定が必要である。

ビット 6 は 8×8 と 8×16 の 2 種類ある CG のどちらを読み込むかの指定である。画面表示には関係しない。ビット 7 はアンダーラインの指定。ここが 1 であり、アトリビュート (漢字 VRAM のビット 5) が 1 なら、(CRTC の設定も許すなら) アンダーラインが表示される。ちなみにアンダーラインの色はグラフィック 1 (青) の色に設定されているものが表示される。BASIC のマニュアルにもあるように、



KSEN 1, 色

PALET 1, 色

は同じ意味を持つ。これはアンダーラインモードを使った後はパレットを再設定すべきであることを示す。また気分が乗れば、PALET 命令の代わりに KSEN 命令を使うのもおつであろう。

### 1FE0H 黒色制御

(IN)/OUT : *turbo*

データ内容	コントロール
ビット 0	黒変換するテキストの色を指定(0~7)
ビット 1	
ビット 2	
ビット 3	テキストの黒変換のON/OFF
ビット 4	グラフィックの黒(透明)を黒変換
ビット 5	グラフィックの青を黒変換
ビット 6	ブランキング期間(枠)を黒変換
ビット 7	未使用

(1=ON, 0=OFF)

チェックポイント グラフィックを黒変換するためには、パレットが0になっていなければならない。また、turbo ではOUT だけだが、ZではINも可能。

黒変換とは早い話がスーパーインポーズ時の黒抜きである。必要のない人には必要ないのだが、欲しい人には盆と正月がいっしょに来たようなものである。

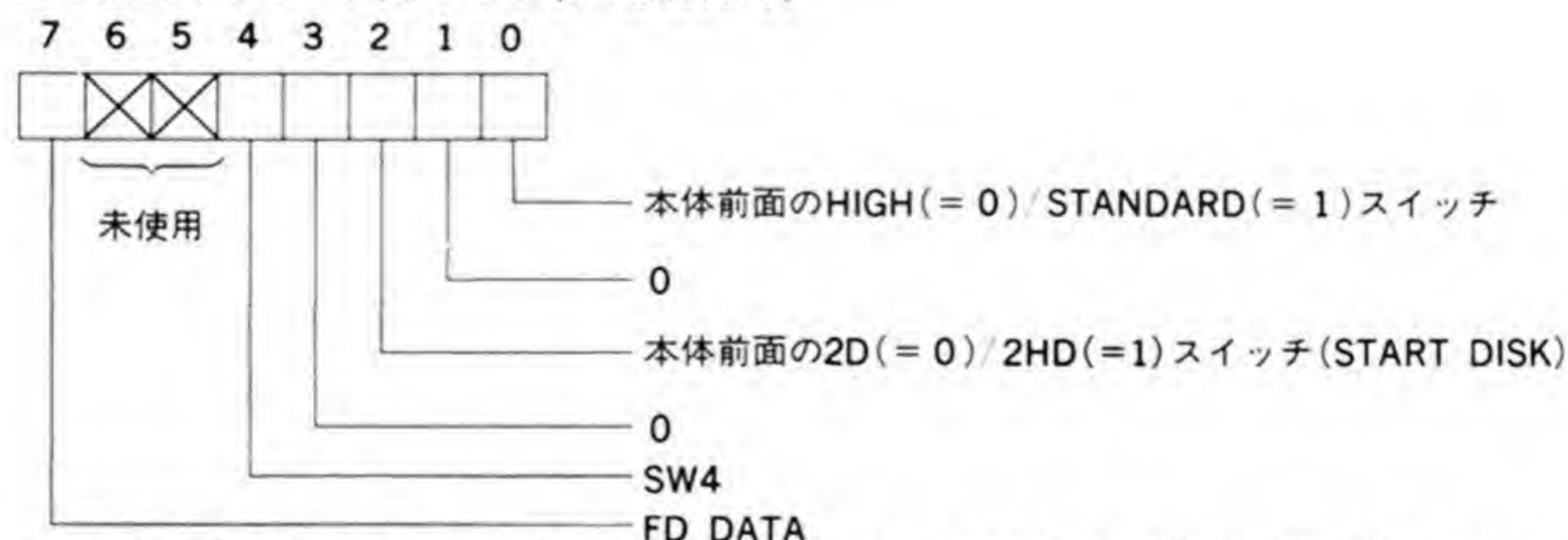
### 1FF0H スタートポート

IN : *turbo* / Z

turboおよび、Zの背面ディップスイッチSW5がOFF=上側のとき



ZでSW5がON=下側のとき(工場出荷時)



チェックポイント ビット1~3はBOOT時のディスクを指定する。SW4は意味を持たない。



No.	SW3	SW2	SW1	セレクト	容 量	記録方式	フォーマット
0	0	0	0	5(3)インチ	320Kバイト	2D : 両面倍密度	
1	0	0	1	5(3)インチ	640Kバイト	2DD : 両面倍密度 倍トラック	
2	0	1	0	5インチ	1 Mバイト	2HD : 両面高密度	
3	0	1	1	5インチ	1 Mバイト	2HD : 両面高密度	IBM
4	1	0	0	8インチ	1 Mバイト	2D256: 両面倍密度	
5	1	0	1	8インチ	1 Mバイト	2D256: 両面倍密度	IBM
6	1	1	0	8インチ	240Kバイト	1S128: 片面単密度	IBM
7	1	1	1	ハードディスク	10Mバイト		

これらのスイッチはただ単に付いているだけである。IPLなどのプログラムがここを見て、その設定にしたがった動作をするわけである。早い話が「フラグ」である。

背面ディップスイッチのSW4は無意味なので、ときにはOFF側に倒して気分を変えるのもよいだろう。

表中の2と3、4と5は同じではない。IBMフォーマットではサイド0、トラック0がFM(単密)記録方式になっている。なお、ビット7の「FD DATA」がその筋である。

ところで初期のころのturbo(市販版かどうかは定かではない)の中には、SW1～SW4の順が逆になっているものもあるらしい。

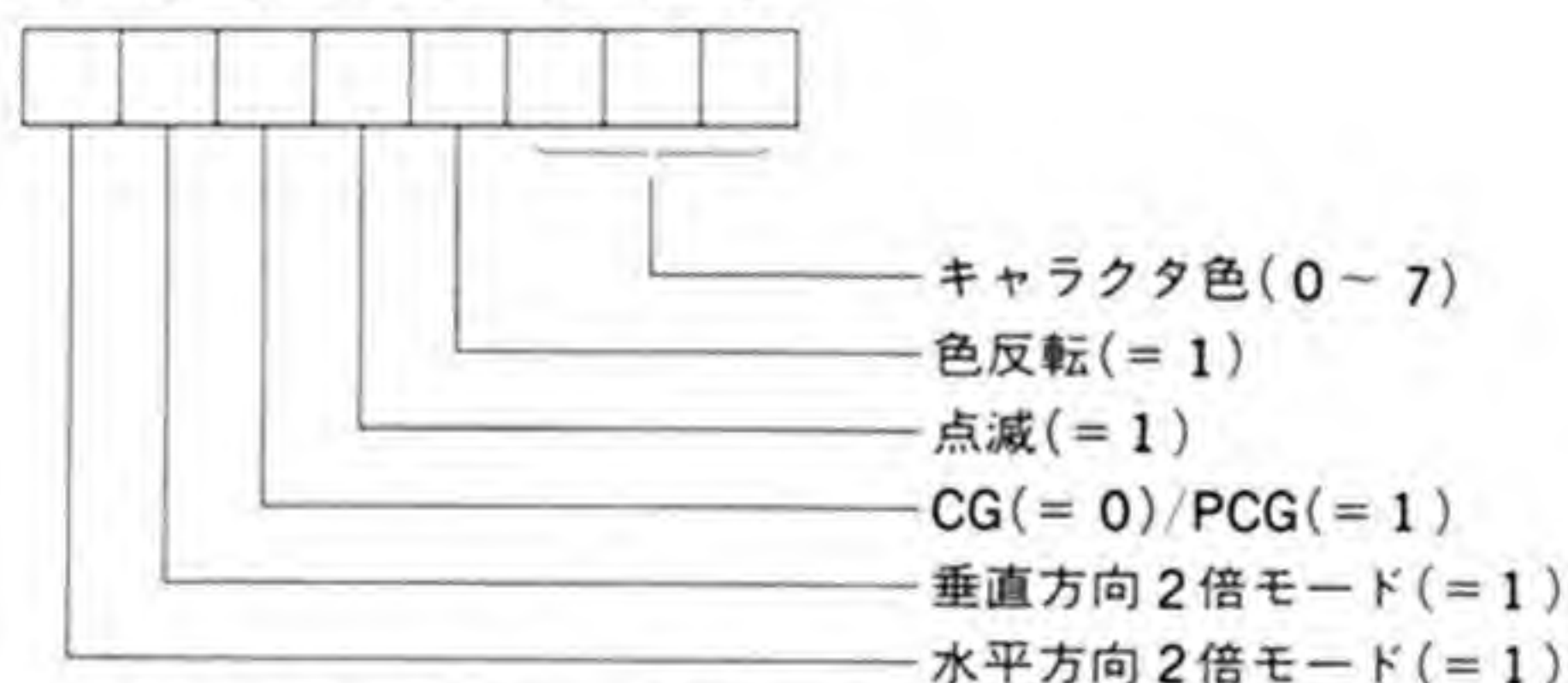
ちなみにZの背面ディップスイッチSW6もディスクに関係している。詳しくはZのユーザーズマニュアルを参照。

## 2000H～27FFH テキストアトリビュート

IN/OUT : 

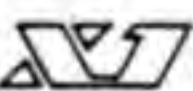
アトリビュート内のビットの意味

7 6 5 4 3 2 1 0

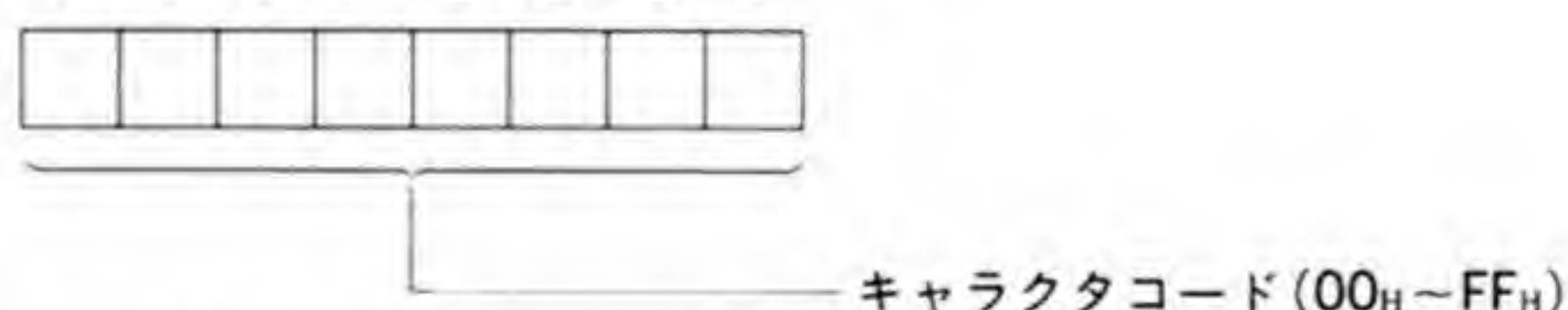


チェックポイント 当たり前のことだが、アトリビュートの内容が0なら色が黒。だから、なにも見えない。Zでは1FB9Hも参照。

## 3000H～37FFH テキストVRAM

IN/OUT :  / **turbo**

7 6 5 4 3 2 1 0

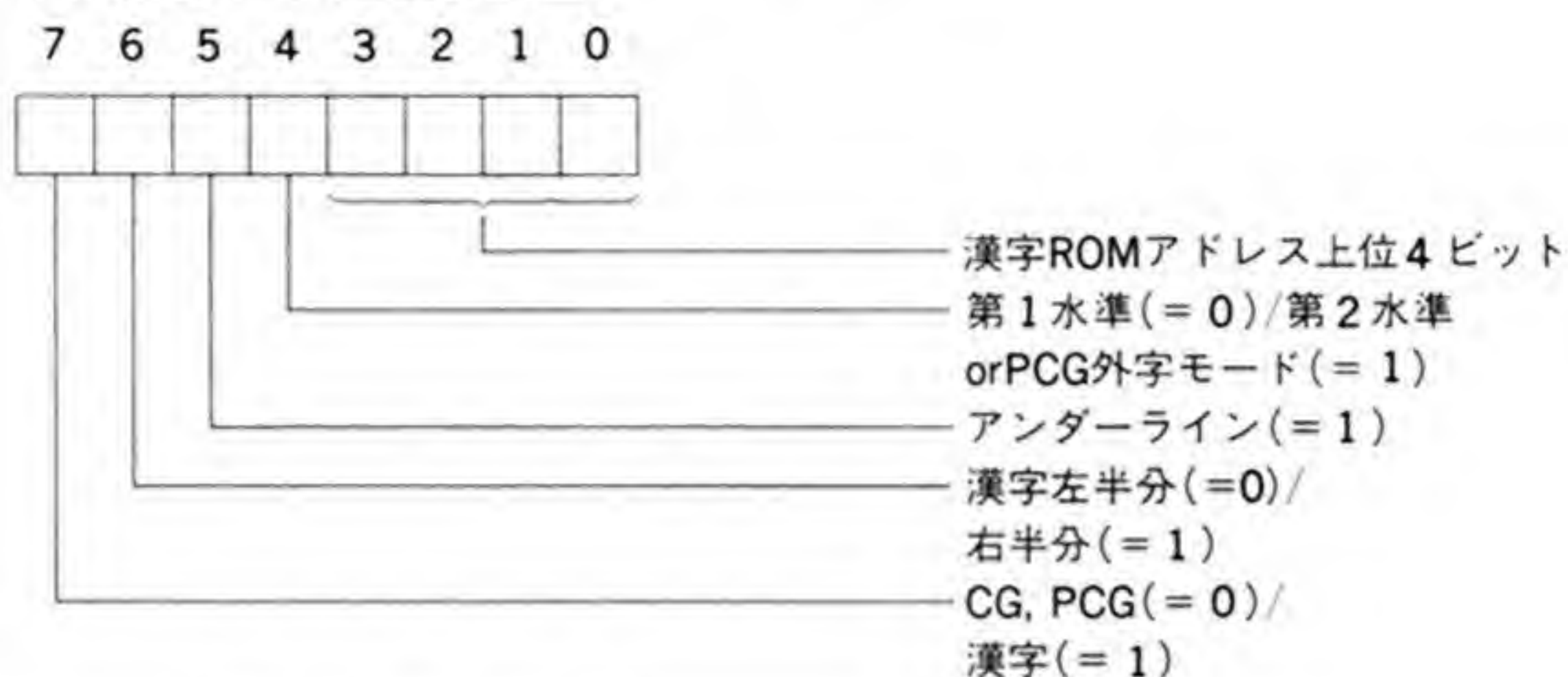


チェックポイント turboで漢字を表示する場合は、漢字ROMアドレスの下位8ビット。



### 3800<sub>H</sub>～3FFF<sub>H</sub> 漢字VRAM (→第3章)

IN/OUT : **turbo**



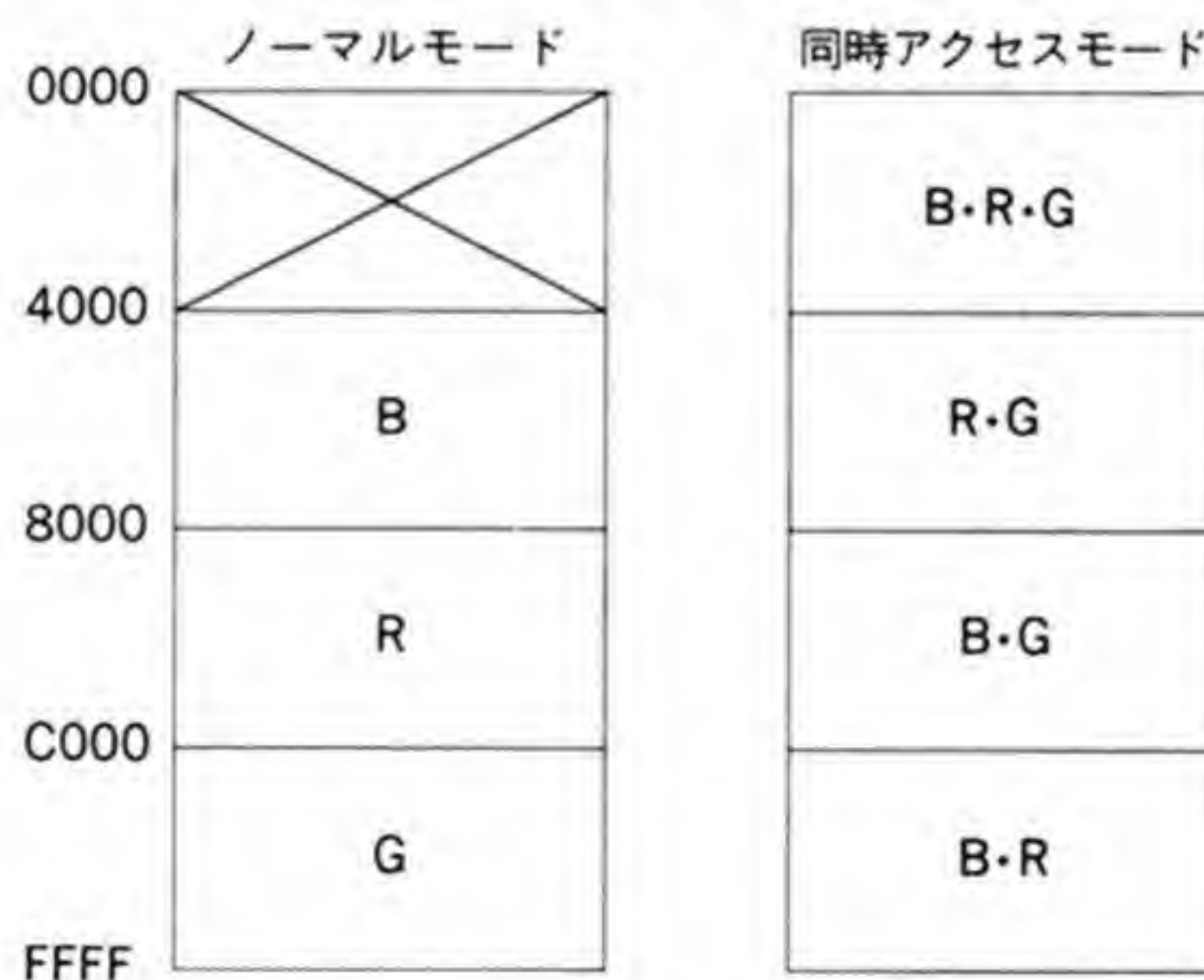
チェックポイント 下の表を見よ。

テキストアトリビュート ビット5= $\overline{\text{ROM}}/\text{RAM}$	漢字VRAM		表 示
	ビット7= $\overline{\text{CG}}/\text{漢字}$	ビット4= $\overline{1}/2$ 水準	
0	0	無関係	CG
0	1	0	漢字(第1水準)
0	1	1	漢字(第2水準)
1	0	0	PCG(ノーマル)
1	0	1	PCG外字①
1	1	無関係	PCG外字②

PCG 外字モードというのは、2 個の PCG を使って  $8 \times 16$  のキャラクタを構成するものである。4 個の PCG で外字一つであるから turbo では計 64 個の外字 (8 色入り) を使えることになる。外字の出し方には 2 とおりあることが表からも分かるが、これは BASIC 上で、全角 ( $16 \times 16$ ) と半角 ( $8 \times 16$ ) の二つを区別するために作られたものだそうである。

### 4000<sub>H</sub>～FFFF<sub>H</sub> グラフィックRAM

IN/OUT : / **turbo**



チェックポイント 同時アクセスモードは1A02<sub>H</sub>, turboでのバンク切り換えは1FD0<sub>H</sub>。



X1 のグラフィックマップは少々複雑であるが、次の式で BLUE 上のアドレス (4000<sub>H</sub> ~ 7FFF<sub>H</sub>) が計算できる。

X=X 座標, Y=Y 座標として 640×200 のとき,

$$\&H4000 + (X \div 8) + ((Y \text{ AND } 7) * 2^{11}) + (Y \div 8) * 80$$

320×200 の場合は最後の “\* 80” を “\* 40” に変えればよい。ここ以外はシフトで計算できるからマシン語で組むのは簡単である。

X1 turbo の 400 ラインではバンク 0 とバンク 1 が 1 段おきになっていて、鋭く上位コンパチを保っている。400 ラインの場合のアドレス計算は、Y AND 1 をバンク No. とし、Y=Y÷2 として、前述の値を計算すればよい。

また、Z では 4096 色モード、64 色モードが加わっているが、その色は、ページ 0, 1, 2, 3 の順に薄い色になっている (ページとは、SCREEN, GRAPH 命令で指定するやつである)。詳しくは第 14 章を参照。





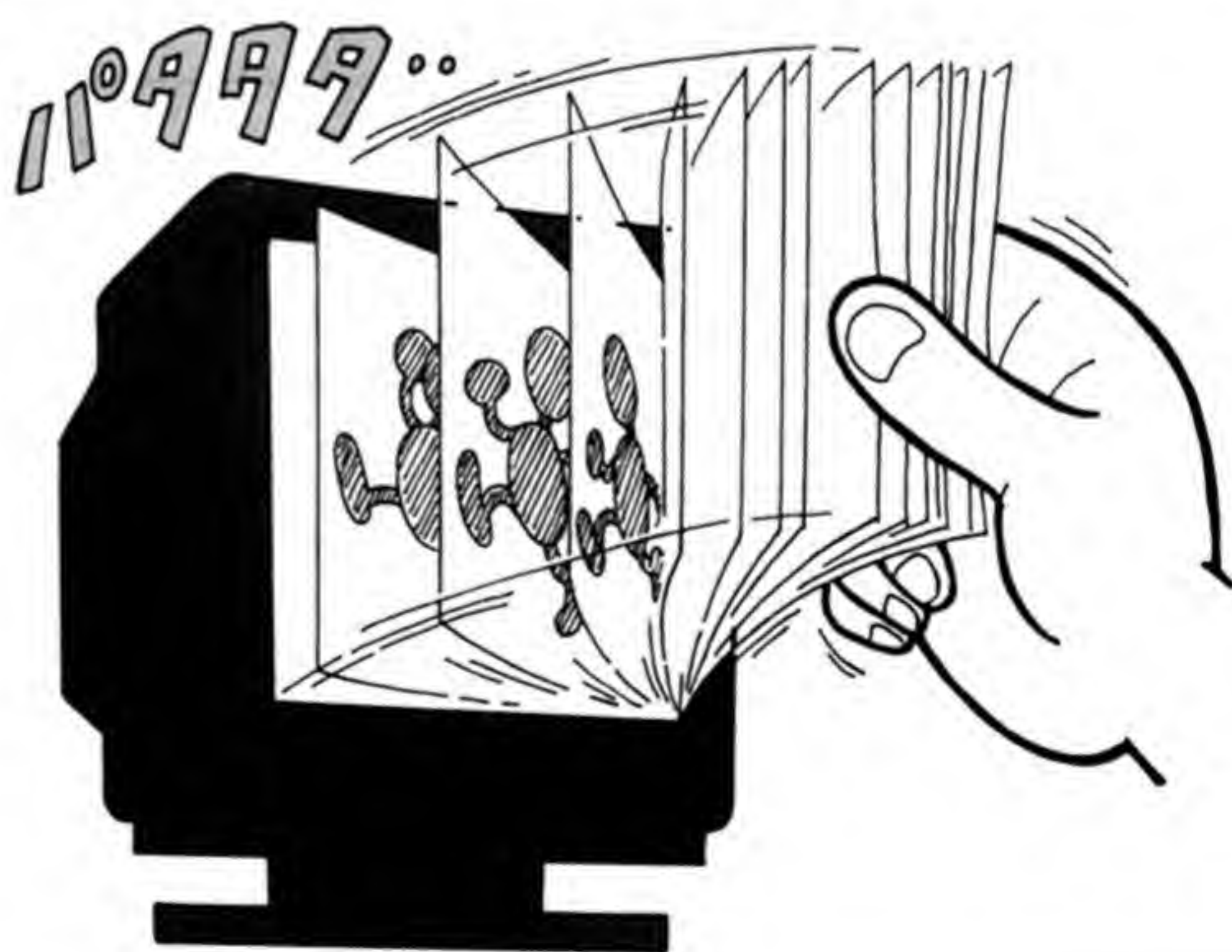


第

1

章

# CRTC



CRTCでどすこいである



## ■ 第1章

# CRTCでどすこいである.....

この章ではCRTCをまな板に上げるのである。CRTCというのは、CRT Controller, すなわち CRT=画面表示をコントロールする LSI のことなのである。つまりはキャラクターやグラフィックやらを、CRT に送って表示する役割をしているのである。

まずは、CRTC がどのようなものかということを目で見るために、リスト 1-1 を実行していただきたい。突如現れる十文字である。しかも微妙なぼかしに、そこはかたない風情を感じるのは私だけではないだろう。このプログラムのネタは単純である。SCREEN コマンドを使って画面に表示するページを、ページ 0 とページ 1 の間で高速に切り換えているために、二つの画面が重なって見えてしまうのである。ほら、何枚かの紙に少しずつ違う絵を描いて、パラパラとやると動いているように見えるというアニメーションの基本があるが、あれを二つの画面（ページ）でやっているのである。

リスト 1-1 ぼかし十文字

```
100 WIDTH 40:INIT
110 GRAPH 0,0:CLS 4
120 PRINT STRING$(40,"-")
130 GRAPH 1,1:CLS 4
140 PRINT STRING$(40,"|")
150 GRAPH 0,0:GRAPH 1,1:GOTO 150
```

で、「BASIC でこのようなことができるのであれば、機械語を使えばもっと美しいことになるのではないか」と思う人が少なからずいるであろう。しかし、世の中はそう甘くないということをリスト 1-2 で噛みしめていただきたい。

リスト 1-2 ぼかし十文字機械語版

```
100 WIDTH 40:INIT
110 GRAPH 0,0:CLS 4
120 PRINT STRING$(40,"-")
130 GRAPH 1,1:CLS 4
140 PRINT STRING$(40,"|")
150 '
160 CLEAR &HFE00:A=&HFE00
170 READ D$:IF D$="END" THEN 190
180 POKE A,VAL("&H"+D$):A=A+1:GOTO 170
190 CALL &HFE00
200 '
210 DATA 01,01,1A      : ' LD BC,1A01H;8255
220 DATA D9            : ' EXX
230 DATA 01,00,18      : ' LD BC,1800H
240 DATA 3E,0C          : ' LD A,0CH
250 DATA ED,79          : ' OUT (C),A ;SET AR
260 DATA 0C             : ' INC C ;BC=1801H
266                     : ' ;
270 DATA CD,1B,FE       : ' LOOP: CALL EDGE
280 DATA AF             : ' XOR A
```



285 DATA ED,79	:	:	OUT	(C),A
290 DATA CD,1B,FE	:	:	CALL	EDGE
300 DATA 3E,04	:	:	LD	A,04H
305 DATA ED,79	:	:	OUT	(C),A
310 DATA 18,F1	:	:	JR	LOOP
320 'FE1B	:	:	:	:
330 DATA D9	:	:	EDGE:	EXX
340 DATA ED,78	:	:	EDGE1:	IN A,(C)
350 DATA F2,1C,FE	:	:	JP	P,EDGE1
360 DATA ED,78	:	:	EDGE2:	IN A,(C)
370 DATA FA,21,FE	:	:	JP	M,EDGE2
380 DATA D9	:	:	EXX	
390 DATA C9	:	:	RET	
400 DATA END	:	:		

リスト 1-2 の機械語ルーチンでは、垂直帰線期間のエッジ（立ち下がり）を検出し、ページの切り換えを行なっているのである（第 0 章の 8255 ②のポート B を参照）。もちろんそんなものを検出せず、機械語まかせに「たたたた」とページ切り換えをしてもいいわけであるが、そうするともっと美しくないことになる（プログラムを変更できる人はやってみることをお勧めする）。それはともかくリスト 1-2 では機械語ルーチンの中に入った後は無限ループになっているので、止めるときはリセットスイッチ（NMI の方）を押していただきたい。それで結果であるが、BASIC でやったのと似たようなものである。すなわちこの「忍法二重画面」はあくまで「こんなこともできる」というだけなのである。

具体的な解説を行なう。表示画面の切り換えは、CRTC のレジスタの書き換えで行なうのである。すなわち表示開始アドレス（CRTC にとっての VRAM 上のアドレス）を変えてやるのである。しかし、いきなりそんなことを言われても理解不能であろう。それが普通なのである。よって、まずは CRTC の基本からやってみたいと思うのである。

## HD46505-SPなめだ

表 1-1 を見ていただきたい。これは X1 に使われている CRTC, HD46505-SP のレジスタ機能表である（→参考文献 3）。この CRTC は 18 個（R0～R17）までのレジスタを持ち、これらのレジスタに値を設定してやる（書き込んでやる）ことでいろいろな画面表示を実現するのである。表の右端には、普通の 40/80 文字モードにおける標準的な設定値を示しておいたので、図 1-1 と併せて見ていただきたい。ただし、これは低解像度（200 ライン）の場合だから、X1 でならよいのだが、turbo BASIC（CZ-8FB02）で高解像度モード（400 ライン）を使っている場合ではちょっと都合が悪いので、その場合はすみやかに CZ-8FB01（もしくは CZ-8CB01）に立ち上げ直していただきたい。

それでは、できるだけ詳しく CRTC の内部レジスタの解説を行なうのである。なお、カッコの中の数字は「設定できる値の範囲」を示している。

### AR（0～17）

まずは隠れレジスタとして AR（アドレスレジスタ）がある。これは、内部レジスタ（R0～R17）のうち、どれにアクセスするかを指定するためのものである（第 10 章の PSG へのアクセスと似ている）。一度設定すれば再設定しない限り AR は同じ値を保持している。



表 1-1 CRTC (HD46505-SP) 内部レジスタ機能表

レジスタ番号	内 容	機 能	40桁	80桁
R0	水平総文字数(-1)	水平走査の周期を文字数に換算して指定する。	55	111
R1	水平表示文字数	1行に表示する文字数を指定する。	40	80
R2	水平同期位置(-1)	水平同期信号を出力する位置(文字数)を指定する。	45	89
R3	同期パルス幅	下位4ビットは水平同期信号のパルス幅を指定。 (文字数に換算) 上位4ビットは垂直同期信号のパルス幅を指定。 (水平走査線本数=ラスタ本数に換算)	52 (34 <sub>H</sub> )	56 (38 <sub>H</sub> )
R4	垂直総文字数(-1)	垂直走査の周期を文字数に換算して指定する。	31	31
R5	総ラスタ調整	垂直走査の周期を微調整する。ラスタ数を指定する。	2	2
R6	垂直表示文字数	表示する文字行数を指定する。	25	25
R7	垂直同期位置(-1)	垂直同期信号を出力する位置(文字数)を指定する。	28	28
R8	インタレース, スキュー	ラスタスキャンモード指定, DISPTMG信号, CUDISP 信号のスキュー(遅れ)を指定する。	0	0
R9	最大ラスタアドレス	1行を構成するラスタ数(走査線)から1引いた値を指定する。	7	7
(R10) (R11)	カーソルスタートラスタ カーソルエンドラスタ	カーソル形状, 表示モードの指定。 X1では使われていない。	— —	— —
R12 R13	スタートアドレス上位 スタートアドレス下位	座標(0,0)に対応するテキストVRAMの値を指定する。 ただしVRAMは0000 <sub>H</sub> ~07FF <sub>H</sub> 番地に対応する。	0 0	0 0
(R14) (R15)	カーソルアドレス上位 カーソルアドレス下位	カーソルの表示アドレスを指定する。 X1では使われていない。	— —	— —
(R16) (R17)	ライトペン上位 ライトペン下位	ライトペン検出アドレスを記憶する。 X1では使われていない。	— —	— —

1800<sub>H</sub>番地に 0 ~17 を OUT すれば AR に値を設定できる。その後, 1801<sub>H</sub>に OUT すれば R0~R17 に値を設定できるわけだ。

**R0 (0 ~255)**

**R1 (0 ~255)**

**R2 (0 ~255)**

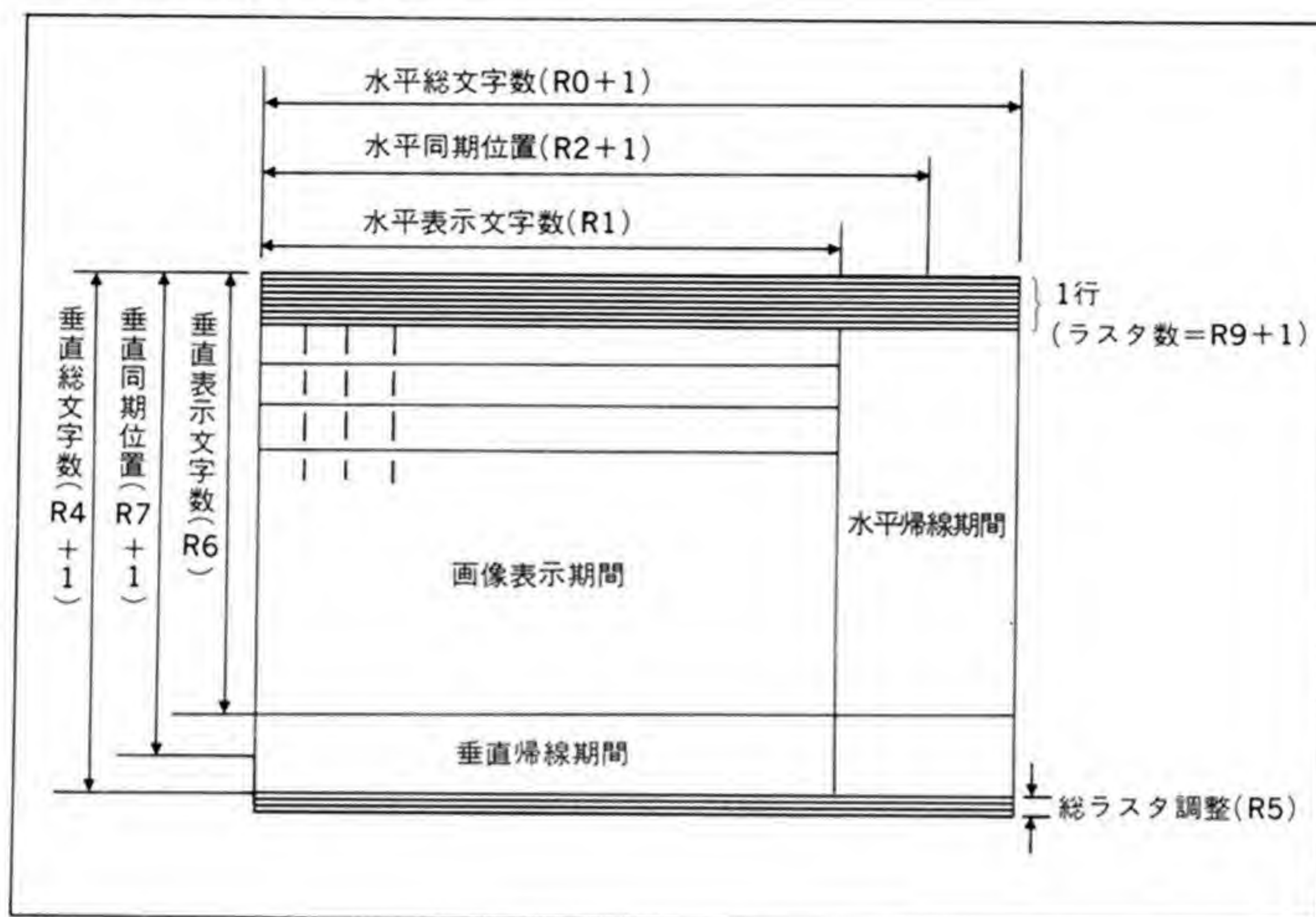
R0 (水平総文字数) は水平走査周期を文字数に換算したものから-1 したものである。総文字数というのは「実際に表示されている1行の文字数」+「水平帰線時間を文字数に換算した値」のことである。R1 は表示文字数。これが「実際に表示される1行の文字数」である。R2 は水平同期位置から-1 したもの(これも文字数に換算する)。

**R3 (0 ~255)**

この1バイトは上位4ビットと下位4ビットの二つに分かれて意味を持つ。下位4ビットは水平同期信号のパルス幅を文字数に換算したもの。X1 では40桁で「4」, 80桁で「8」を使っている。上位4ビットは垂直同期パルス幅をラスタ数に換算したものである。同様



図 1-1 CRTC 内部レジスタの意味



に X1 では低解像度モニタ（標準解像度モニタ）に対しては「3」、turbo での高解像度モニタに対しては「8」を指定する。ところでいきなり「ラスタ」という言葉が出てきたが、これは「走査線」のことと思って差し支えない。ほら、200 ラインのディスプレイをよく見ると 200 本分の横縞が見えるであろう。アレである。

**R4 (0～127)**

**R6 (0～127)**

**R7 (0～127)**

R4 は R0, R6 は R1, R7 は R2 と対応している。すべて文字数に換算して設定する。

**R5 (0～31)**

これは垂直方向の帰線周期の微調整である。R4 は文字数だが、R5 はラスタ本数、すなわち走査線の本数で指定する。このレジスタはスーパーインポーズ時のスムーズスクロール (BASIC のスクロール命令) と関係している。第 0 章の 8255 ②: 1A02<sub>H</sub> の説明を参照のこと。

**R8**

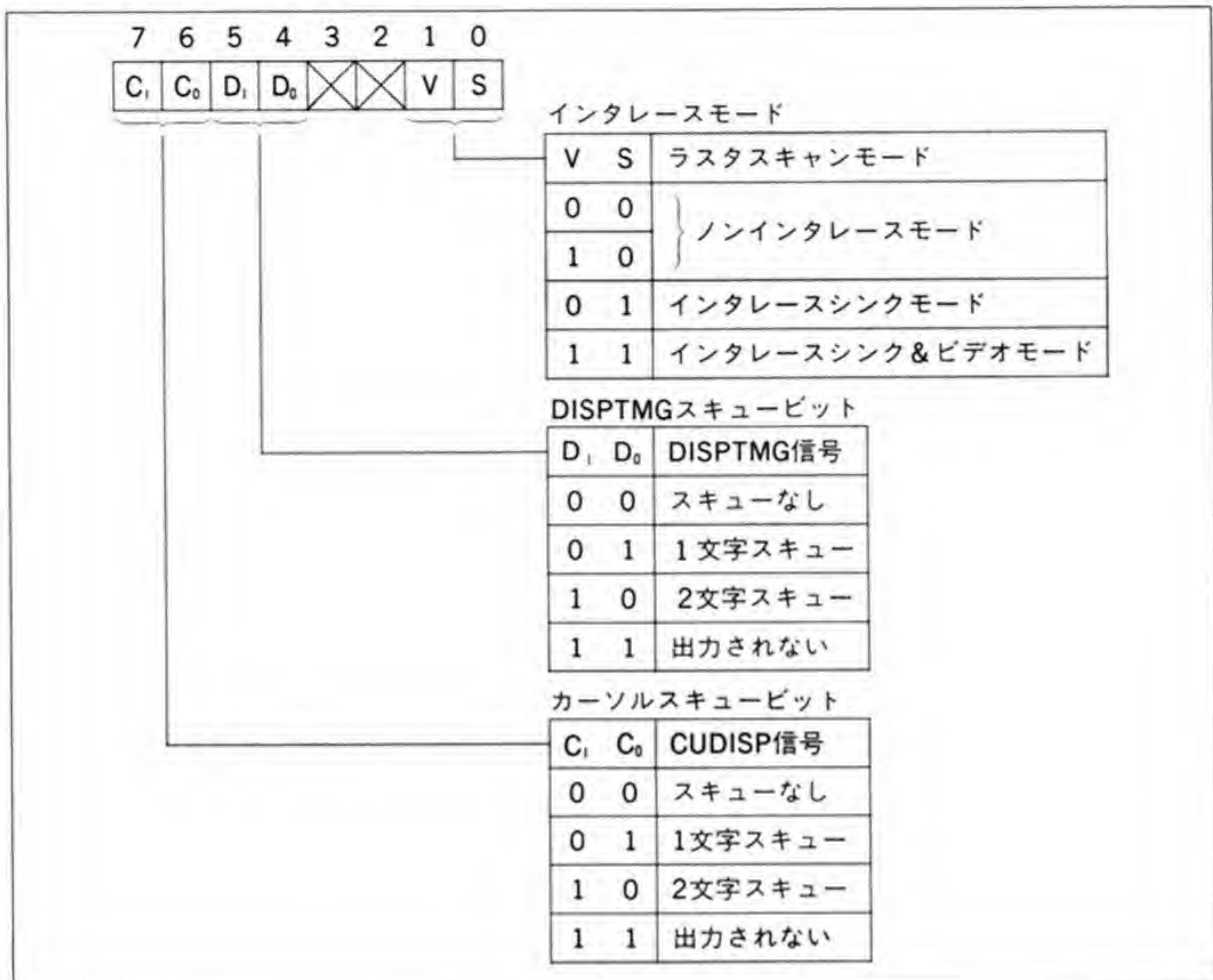
表 1-2 から分かるように、ビットごとに別々の意味を持つ。X1 を普通に使う場合には常に 0 でよいが、それ以外の値を設定することも可能である。

**インタレースモード (ビット 1, 2)**

普通はノンインタレースモードである。インタレースシンクモードにすると画面が微妙に震えて、目を悪くするにはうってつけである。インタレースシンク&ビデオモードにすると、画面は縦にぐしゃっと潰れる（本当は潰れるだけではない）。そして、R5 を調節しな



表 1-2 R8 の機能



いと画面が上下にくるくる流れるであろう（垂直同期がずれるのだ）。

#### DISPTMG 信号（ビット 5， 4）

普通はスキューなしのモードが設定されている。スキューを指定した場合は左端の 1 文字、もしくは 2 文字が CRT の右端に行ってしまう。その他の文字は移動しない。「11」で「出力されない」を指定すると、画面には何も現れなくなる。

#### CUDISP 信号（ビット 7， 6）

X1 ではこの信号はオープン（無接続）になっているので、ここをいじっても何も起こらない。同じ CRTC を使っていた PASOPIA7 ではこの機能を使ってカーソルを出していた。

#### R9（0～31）

最大ラスタアドレスというのは、「1 文字を何ラスタで表示するか」なのだ。R9 にはその数 - 1 を設定する。X1 では文字は 8 ラスタで表示されるから 7 を指定する。turbo では 7 か 15、そしてアンダーラインモードのときは、アンダーラインの分も入れるので、9 か 19 となる。

#### R10（0～127），R11（0～31）

X1 では一切使っていない。カーソルの形状、モードの指定に使われる。

#### R12（0～63）

#### R13（0～255）



二つのレジスタを合わせた 14 ビットで表示開始アドレス（左上端に表示するキャラクタのテキスト VRAM のアドレス）を指定する。X1 では 2048 バイトの RAM（キャラクタ表示用）しか持っていない。これは 2 進数では &B000000000000 ~ &B111111111111 (11 桁) であるから X1 では R12 は下位 3 ビットだけに意味がある。最初に例を挙げたリスト 1-1 の画面切り換えは、R12, R13 をいじっている。

#### R14 (0 ~ 654), R15 (0 ~ 255)

カーソルを表示するアドレスを指定する。X1 では使われていない。

#### R16 (0 ~ 64), R17 (0 ~ 255)

ライトペンの位置を読み取るためのレジスタである。X1 では使われていない。

ここで全体的な CRTC の解説の補足しておく。

- 1) HD46505-SP の仕様では、R0~R11 が書き込み可能、R12~R15 が書き込み/読み込みの両方が可能、R16, R17 が読み込みだけ可能となっている。ところが X1 では読み込みは一切できない。
- 2) CRTC の説明では「文字」で機能を説明した。グラフィックは、「同じ位置に表示される文字」と同様な扱いを受ける。つまり、スタートアドレスを変えたならば、グラフィックも同様に表示が移動する。たとえば、テキストの 3000<sub>H</sub> 番地と、G-RAM の 4000<sub>H</sub>, 4800<sub>H</sub>, 5000<sub>H</sub>, 5800<sub>H</sub>, 6000<sub>H</sub>, 6800<sub>H</sub>, 7000<sub>H</sub>, 7800<sub>H</sub> の 8 バイトは、CRTC をいかに設定しようとも重なり合って表示される。

以上である。ここらで気分転換にリスト 1-3 を打ち込んで遊んでいただきたい。このプログラムは R12, R13 をいじってグラフィックを上下左右に 8 ドット単位でスクロールさせるものである。VRAM には触らずに、CRT 上での表示位置（表示開始アドレス）を動かすことによってスクロールを実現している。念のために注意しておくが、これは「見て面白い」のであって、あまり実用にはならない。また、時々ちらつきが出るが、それは垂直帰線期間の間（つまり画面に何も表示されていない間）を考慮せず、R12, R13 を設定しているからである。その気のある人はリスト 1-2 を参考にして正しく機械語で組んでみていただきたい。

#### リスト 1-3 ハードウェアスクロールもどき

```
100 INIT:WIDTH 80:CLS 4:WIDTH 40
110 GRAPH 0,0
120 FOR J=1 TO 7:CIRCLE (280,160),J*5,J:NEXT
130 Y= 0:FOR X= 0 TO 30          :GOSUB "START":NEXT
140 X=30:FOR Y= 0 TO 15          :GOSUB "START":NEXT
150 Y=15:FOR X=30 TO 0 STEP -1:GOSUB "START":NEXT
160 X= 0:FOR Y=15 TO 0 STEP -1:GOSUB "START":NEXT
170 GOTO 130
180 '
190 LABEL "START"
200 A=X+Y*40:A1=A ¥ 256:A2=A MOD 256
210 'GRAPH 0,0
220 OUT &H1800,12:OUT &H1801,A1
230 OUT &H1800,13:OUT &H1801,A2
240 RETURN
```



## さらにCRTCを究める

リスト 1-4, 1-5 は画面モードのサンプルである。

リスト 1-4 X1 用画面サンプル

```
100 INIT:CIRCLE(200,100),90
110 RESTORE "4025L"
120 FOR Z=1TO 8:GOSUB"MAIN":BEEP:WHILE(INKEY$(0)=""):WEND
130 NEXT:END
140 '
150 LABEL"FOO":WIDTH80:RESTORE"8025L":GOSUB"MAIN":BEEP
160 INPUT A$:ON ERROR GOTO 190:RESTORE A$:ON ERROR GOTO 0
170 GOSUB"MAIN":END
180 'ERROR TRAP
190 PRINT "ILLIGAL LABEL":BEEP:END
200 '
210 LABEL "BAR":WIDTH 80:RESTORE "8025L"
220 GOSUB"MAIN":END
230 '
240 LABEL"MAIN"
250 GOSUB"SETCRTC":GOSUB"80/40"
260 RETURN
270 '
280 LABEL"SETCRTC"
290 FORI=0TO13
300 READ D$:D=VAL("&H"+D$)
310 OUT&H1800,I:'SET CRTC REG. No.
320 OUT&H1801,D:'SET DATA
330 NEXT:RETURN
340 '
350 LABEL"80/40"
360 READ D$:D=VAL("&H"+D$)
370 OUT&H1A03,D
380 RETURN
390 '
400 LABEL"4025L":DATA 37,28,2D,34,1F,02,19,1C,00,07,00,00,00,00,0D:'40,25
410 LABEL"4020L":DATA 37,28,2D,34,18,08,14,16,00,09,00,00,00,00,0D:'40,20
420 LABEL"4012L":DATA 37,28,2D,34,0F,02,0C,0E,00,0F,00,00,00,00,0D:'40,12
430 LABEL"4010L":DATA 37,28,2D,34,0B,12,0A,0B,00,13,00,00,00,00,0D:'40,10
440 '
450 LABEL"8025L":DATA 6F,50,59,38,1F,02,19,1C,00,07,00,00,00,00,0C:'80,25
460 LABEL"8020L":DATA 6F,50,59,38,18,08,14,16,00,09,00,00,00,00,0C:'80,20
470 LABEL"8012L":DATA 6F,50,59,38,0F,02,0C,0E,00,0F,00,00,00,00,0C:'80,12
480 LABEL"8010L":DATA 6F,50,59,38,0B,12,0A,0B,00,13,00,00,00,00,0C:'80,10
490 '
500 LABEL"Q8006L":DATA 6F,50,59,38,07,02,06,07,00,1F,00,00,00,00,0C:'80,06
510 LABEL"Q8005L":DATA 6F,50,59,38,06,02,05,05,00,27,00,00,00,00,0C:'80,05
520 LABEL"Q8005D":DATA 6F,50,59,38,06,04,05,05,00,2F,00,00,00,00,0C:'80,05
530 LABEL"Q8003L":DATA 6F,50,59,38,03,02,03,03,00,3F,00,00,00,00,0C:'80,03
```

リスト 1-5 X1 turbo 用画面サンプル

```
100 INIT:CIRCLE(200,100),90
110 RESTORE "4025L"
120 FOR Z=1TO18:GOSUB"MAIN":BEEP:WHILE(INKEY$(0)=""):WEND
130 NEXT:END
140 '
150 LABEL"FOO":WIDTH80:RESTORE"8025H":GOSUB"MAIN":BEEP
160 INPUT A$:ON ERROR GOTO 190:RESTORE A$:ON ERROR GOTO 0
170 GOSUB"MAIN":END
180 'ERROR TRAP
190 PRINT "ILLIGAL LABEL":BEEP:END
200 '
210 LABEL "BAR":WIDTH 80:RESTORE "8025H"
220 GOSUB"MAIN":END
230 '
240 LABEL"MAIN"
250 GOSUB"SETCRTC":GOSUB"80/40":GOSUB"1FD*"
260 RETURN
270 '
280 LABEL"SETCRTC"
290 FORI=0TO13
300 READ D$:D=VAL("&H"+D$)
```



```

310 OUT&H1800,I:'SET CRTC REG. No.
320 OUT&H1801,D:'SET DATA
330 NEXT:RETURN
340 '
350 LABEL"80/40"
360 READ D$:D=VAL("&H"+D$)
370 OUT&H1A03,D
380 RETURN
390 '
400 LABEL"1FD*"
410 READ D$:D=VAL("&H"+D$)
420 OUT &H1FD0,D
430 RETURN
440 '
450 LABEL"4025L":DATA 37,28,2D,34,1F,02,19,1C,00,07,00,00,00,00,0D,00:'40,25
460 LABEL"4020L":DATA 37,28,2D,34,18,08,14,16,00,09,00,00,00,00,0D,80:'40,20
470 LABEL"4012L":DATA 37,28,2D,34,0F,02,0C,0E,00,0F,00,00,00,00,0D,04:'40,12
480 LABEL"4010L":DATA 37,28,2D,34,0B,12,0A,0B,00,13,00,00,00,00,0D,84:'40,10
490 '
500 LABEL"8025L":DATA 6F,50,59,38,1F,02,19,1C,00,07,00,00,00,00,0C,00:'80,25
510 LABEL"8020L":DATA 6F,50,59,38,18,08,14,16,00,09,00,00,00,00,0C,80:'80,20
520 LABEL"8012L":DATA 6F,50,59,38,0F,02,0C,0E,00,0F,00,00,00,00,0C,04:'80,12
530 LABEL"8010L":DATA 6F,50,59,38,0B,12,0A,0B,00,13,00,00,00,00,0C,84:'80,10
540 '
550 LABEL"4025H":DATA 35,28,2D,84,1B,00,19,1A,00,0F,00,00,00,00,0D,03:'40,25
560 LABEL"4025M":DATA 35,28,2D,84,1B,00,19,1A,00,0F,00,00,00,00,0D,01:'40,25
570 LABEL"4020H":DATA 35,28,2D,84,15,08,14,15,00,13,00,00,00,00,0D,81:'40,20
580 LABEL"4012H":DATA 35,28,2D,84,0D,00,0C,0D,00,1F,00,00,00,00,0D,07:'40,12
590 LABEL"4010M":DATA 35,28,2D,84,0D,00,0C,0D,00,1F,00,00,00,00,0D,05:'40,10
600 '
610 LABEL"8025H":DATA 6B,50,59,88,1B,00,19,1A,00,0F,00,00,00,00,0C,03:'80,06
620 LABEL"8025M":DATA 6B,50,59,88,1B,00,19,1A,00,0F,00,00,00,00,0C,01:'80,06
630 LABEL"8020H":DATA 6B,50,59,88,15,08,14,15,00,13,00,00,00,00,0C,81:'80,05
640 LABEL"8012H":DATA 6B,50,59,88,0D,00,0C,0D,00,1F,00,00,00,00,0C,07:'80,05
650 LABEL"8012M":DATA 6B,50,59,88,0D,00,0C,0D,00,1F,00,00,00,00,0C,05:'80,03

```

まずリスト 1-4 から解説しよう。これは X1 用であるが、当然のことに CZ-8FB01 を使って 200 ラインモードの画面であれば turbo でも使える。RUN させたなら、何かキーを押すたびに 8 とおりの画面モードが出現する。終わったなら、

"BAR ␣

と打ち込むと正常な画面になる。ただし、打ち込み間違いがあると表示がめちゃくちゃになるので、RUN させる前にセーブしておくべきである。

このプログラムの中心は 400~480 行のデータ文である。サブルーチン "MAIN" は 14 個のデータを CRTC の内部レジスタ R0~R13 までに設定する。最後の 0D<sub>H</sub> もしくは 0C<sub>H</sub> は 8255 に OUT されるデータである。これはビットセット/リセットモードを使っている。I/O の 1A02<sub>H</sub> 番地のビット 6 を操作しているのだ。第 0 章の I/O マップを参照していただきたい。

X1 における正常な 40 桁、80 桁の画面はそれぞれ "4025L" と "8025L" に対応している。だから、それ以外のモードのときは実に楽しくシュールである。

これ以外の使い方としては、「"FOO ␣」と打ち込むと、画面に「？」が出て入力モードになるはずである。このときラベル名を入れるとそのラベルの直後にあるデータを使って画面モードを設定することができる（たとえば ?Q8006L ␣）。このときは 500 行以降のデータも使えることになる。シュールさを味わって欲しい。ただし R5 の値が微妙であるから、場合によっては垂直同期などがずれて、表示が流れる可能性がある。いろいろ試してみるのも面白いかもしれない。

次に turbo 用のリスト 1-5 である。こちらの方は、1 行中のデータが一つ増えている。



これは 1FD0<sub>H</sub> (画面管理ポート) への出力データである。問題になるのはビット 0～2 と 7 である。表 1-3 を見て納得していただきたい。

以上が基礎編である。

表 1-3 画面モードと画面管理ポート(1FD0<sub>H</sub>)

	低 解 像 モ ー ド				高 解 像 モ ー ド				
	25行	12行	20行	10行	25行	25行	12行	12行	20行
テキスト	25行	12行	20行	10行	25行	25行	12行	12行	20行
グラフィック	200ライン	192ライン	なし	なし	200ライン	400ライン	192ライン	384ライン	なし
ビット0	0	0	0	0	1	1	1	1	1
ビット1	0	0	—	—	1	0	1	0	—
ビット2	0	1	0	1	0	0	0	1	0
ビット7	0	0	1	1	0	0	0	0	1

## もっと表示するのである

X1 で表示できる文字数は、最大で 80×25 の 2000 文字であると一般に信じられている。しかし、その認識は甘いのである。次のプログラム、リスト 1-6 は 81×25、82×25 の画面モードを作ってしまうのだ。それぞれの場合について、よく文字数を数えてみていただきたい。最後の 1 行は途中までしか表示していないが、これは BASIC 上でやっているの、勝手にスクロールさせないためなのだ。アドレスを計算して POKE@ や OUT 文を使えばきっちり表示する (その場合はアトリビュートの設定も忘れずに)。

リスト 1-6 WIDTH 81

```

100 KEY 2, "WI, 80" + CHR$(13)
110 L=81: ' OR L=82
120 INIT:WIDTH 80:CLS 4
130 OUT &H1800,1:OUT &H1801,L
140 FOR I=1 TO 24:PRINT STRING$(L,CHR$(64+I));:NEXT
150 PRINT STRING$(80-(L-80)*25-1,CHR$(64+25));
160 GOTO 160

```

[F2] には "WI, 80" を定義してあるから、まともな画面に戻すときに使って欲しい。ただし turbo の場合は、CZ-8FB01 を使うこと。それから、CRT の調節の具合によっては、右端が画面からはみ出してしまふことがあるかもしれない。そのようなときは素直に納得していただきたい。

実行してみると分かるが、L=82 で 1 行 82 文字にしたとき、右下に「!」が 2 個表示される。これはなぜかという、82×25=&H802=&H800+2 だからなのである。すなわち、CRTC は 2050 個の文字を表示しようとして、ともかくも 2048 個目まではどっど表示するのである。ところがどっこい、テキスト VRAM はそこで終わりなので、CRTC は仕方なくテキスト VRAM の先頭に戻って 2 文字分をもう一度表示し、その場をつくるのである。そう、CRTC はどんな挑戦でも受けているのである。



## 全画面である

最後にビデオマニアのために全画面ワイプ用のプログラムをリスト 1-7 に示しておく。これは turboZ のオーバースキャンみたいなものと考えて差し支えないであろう。しつこいようだが turbo BASIC では動かないので、CZ-8FB01 を使っていただきたい。

リスト 1-7 全画面ワイプ

```
100 GOSUB"BAR":' INITIALIZE
110 GOSUB"PCGDEF":' PCG DEFINITION
120 WIDTH80:CLS4:INIT
130 RESTORE"ALLWP":GOSUB"MAIN":' SET CRTC
140 CGEN1:COLOR7:'WIPE COLOR = 7 (WHITE)
150 FOR I=0 TO 8:CLS,I:PAUSE1:NEXT
160 FOR I=8 TO 0 STEP -1:CLS,I:PAUSE1:NEXT
170 IF INKEY$(0)=" " THEN 150
180 GOSUB"BAR":END:' CRT=NORMAL AND END
190 '
200 LABEL"BAR":INIT:CLS,32:WIDTH40:RESTORE"4025L"
210 GOSUB"MAIN":RETURN
220 '
230 LABEL"MAIN"
240 GOSUB"SETCRTC":GOSUB"80/40"
250 RETURN
260 '
270 LABEL"SETCRTC"
280 FOR I=0 TO 13
290   READ D$:D=VAL("&H"+D$)
300   OUT &H1800,I:'SET CRTC REG. No.
310   OUT &H1801,D:'SET DATA
320 NEXT:RETURN
330 '
340 LABEL"80/40"
350 READ D$:D=VAL("&H"+D$)
360 OUT &H1A03,D
370 RETURN
380 '
390 LABEL"ALLWP":DATA 37,2E,30,34,1F,02,1C,1E,00,07,00,00,00,00,0D:'46,28
400 LABEL"4025L":DATA 37,28,2D,34,1F,02,19,1C,00,07,00,00,00,00,0D:'40,25
410 '
420 LABEL"PCGDEF"
430 M=&HFF
440 FOR I=0 TO 8
450   P$=STRING$(I,&H0)+STRING$(8-I,M)
460   DEFCHR$(I)=P$+P$+P$
470   M=INT(M/2)
480 NEXT:RETURN
```

プログラムの最初のイニシャライズでCRTCを正常にしているのは、PCG定義をするためである。PCGを定義するためには、CRTCが正しく普通のモードになっている必要があるのだ。詳しくは第2章の PCG を参照していただきたい。さて、プログラム中に

CLS, I

という命令が出てくるが、これは CZ-8FB01 の隠れ命令で、I の値を ASCII コードとするキャラクタをスペースとみなして画面をクリアするのである。一度この命令が実行されると、そのキャラクタが論理上のスペースとみなされるので、普通の女の子に戻ってまともな CLS をしたいときは、

CLS, 32

を実行する必要がある。これは turbo BASIC では削られている機能である。

このプログラムを実行すると、やがて画面は 46 字×28 行モードになる。しかし BASIC は 80×25 のモードのつもりで動いている。これによって「CLS, I」を実行すると、46 文



字×28行をすべてキャラクタ=CHR\$(I)で埋めてくれる。プログラムはキーが押されるまでワイプを繰り返す。何かキーが押されたなら、画面を正常にして終了する。打ち込み間違いをすると画面がぐちゃぐちゃになるので注意すること。なお、このプログラムはスーパーインポーズ用である。スーパーインポーズでないときは画面の下の方に少し隙間ができていたが、スーパーインポーズにすると、びょーんと伸びてくるのだ。これぞまさしく**東洋の神秘**と言わざるを得まい。

この章のサンプルプログラムでは十分に示せたかどうか分からないが、CRTCという石はかなり面白い。たとえば画面を20×25にして4面に分けて使うとかができる。しかし、実際に実用に値する裏技はあまりなく、私が考えついたのはリスト1-7の全画面ワイプぐらいなのである。これは、裏技は所詮裏技にすぎないということなのであろうか？

しかしX1のCRTCはいじろうと思えばいじれるのである。そーなればとことんいじり回すのがその筋のその筋たるゆえんなのである。



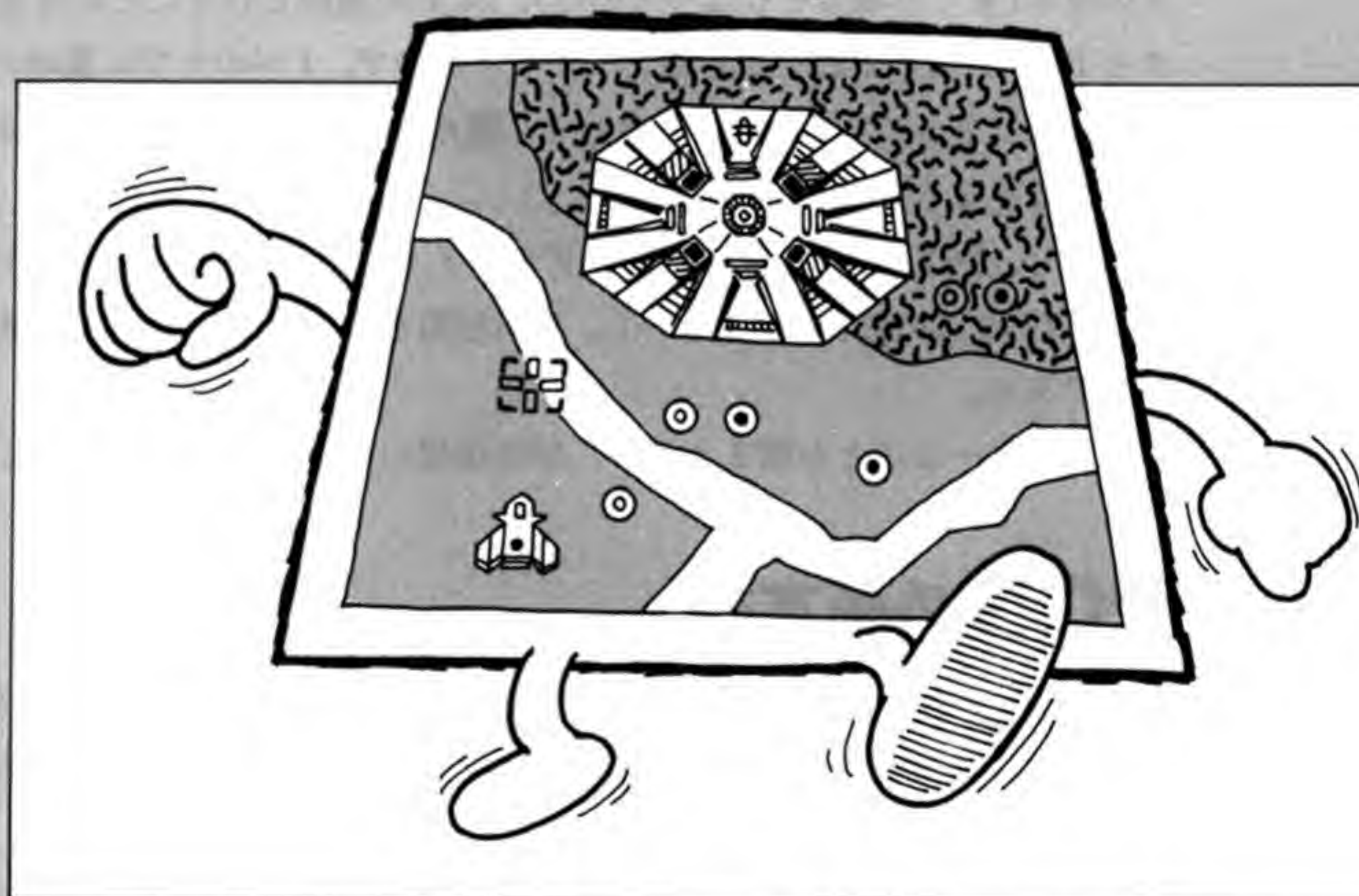
# 第

# 2

# 章

## PCG

あ〜す8



PCGは二度おいしいのである



## ■第2章

# PCGは二度おいしいのである・・・

この章では PCG をやるのである。念のために言っておくと、PCG とは Programmable Character Generator の意味である。ここで言う Programmable とは「書き換えられるよーん」ということ。そして Character Generator とはそのままだと、「文字生成機」ということになる。文字生成機とは実に面妖であるが、つまりは「この ASCII コードの文字は、こーゆーパターンになっているのだ」ということを決めるものなのだ。Programmable でない場合は、そのパターン (FONT: フォント) は ROM になっているわけである。そのような背景があるので、PCG は場合によっては、RAM CG などと呼ばれることもあるのである。その反対は ROM CG である。

さて、周知のよーに X1 や MZ-2500 ではこの PCG がとてもパワフルなので、XEVIOUS やグラディウスなどがバシバシと動くのであった。PCG の利点は、(アトリビュートも入れるとすると) 2 バイトのデータを転送するだけで、8 ドット×8 ドット×8 色のパターンを移動させることができる、ということにある。グラフィックでやるなら、24 バイトの転送であるから、単純に考えれば 12 倍の速度である。もし、アトリビュートを移動しなくてよいなら(多くの場合そうになっている)、24 倍の速度ということになる。しかもグラフィックと PCG の重ね合わせが自由だったりするので、1 バイトで二度おいしいのである。ゲーム製作者にとっては、これはまさしく、棚から PCG、もしくは、鴨が PCG を背負ってきた、と言えよう。

しかし、PCG といえども完全にグラフィックと置き換えられるわけではない。すなわち、PCG は 256 個しか使えないし、表示位置も 8 ドット単位でしか指定することができないのである。

以上のよーなことを踏まえつつ、話は佳境に入ってゆくのであった。

## PCGの基本技

BASIC 上で PCG を使うには、まずは PCG を定義しなければならない。その後でモードを切り換えて表示するのである。すなわち、CZ-8FB01, 8CB01 では、

```
10 INIT:CLS
20 DEFCHR$(77)=24 バイトの文字列
30 CGEN 1
40 PRINT CHR$(77)
```

である。当たり前であった。

これを機械語的にやると、30~40 行は大体、

```
30 OUT &H2800, &H27
```

44 試験に出る X1

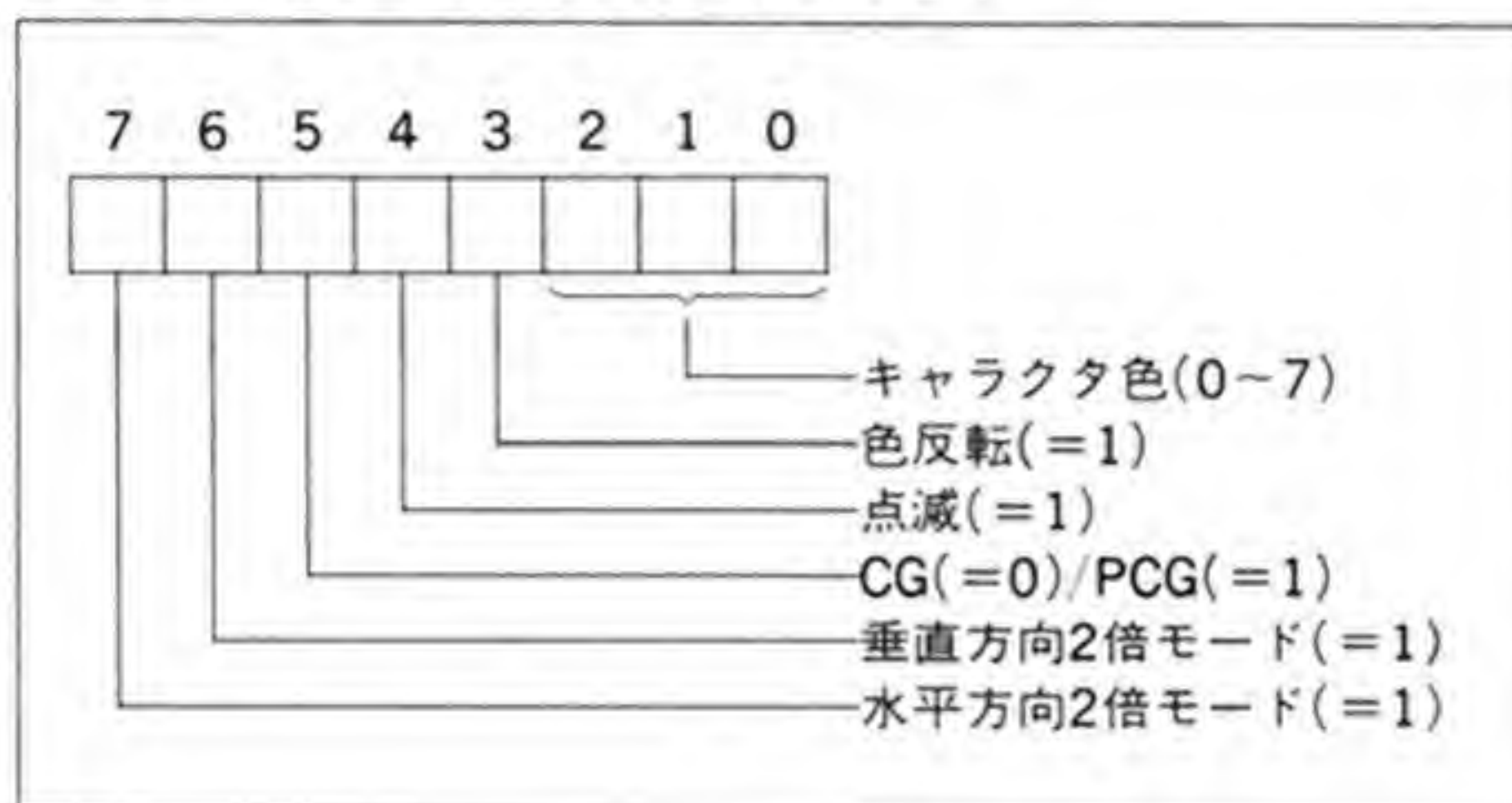


## 40 OUT &H3000, 77

となる。やっていることは完全に同じではない点に注意していただきたい。

さて、30 行の OUT 文はアトリビュートの設定である。アトリビュートの意味は第 0 章にも載せたが、ここでも図 2-1 に示しておく。これにより、&H27 とは色が 7 であり、かつ PCG であるということが判明するであろう。40 行の 77 は単なる ASCII コードなわけだ。

図 2-1 アトリビュート内のビットの意味



アトリビュートを他のものにすれば、それぞれ横 2 倍、縦 2 倍、反転などなどになる。カラーを 7 以外にすると、PCG の色が変わる。たとえばリスト 2-1 のように ASCII コード 77 の PCG に、

**青は A, 赤は B, 緑は C**

と定義しておいて、色を 0~7 まで変化させて表示すると、それなりのパターンが表示される。試していただきたい。

リスト 2-1 PCG とアトリビュートのカラー

```
100 WIDTH 40
110 A$=""
120 FOR I=&H41 TO &H43
130   A$=A$+LEFT$(CGPAT$(I),8)
140 NEXT
150 DEFCHR$(77)=A$
160 FOR I=0 TO 7
170   OUT &H3000+I,77
180   OUT &H2000+I,&H20+I
190 NEXT
200 LOCATE 0,5
```

## PCGの定義

BASIC では、PCG の定義には DEFCHR\$文を使うわけである。turbo では問題はないのだが、X1 では DEFCHR\$は異常に遅い命令として津々浦々に知れ渡っている。以下に、なにゆえにそのように遅いのかを示しつつ、超高速 PCG 定義などへと到達する予定である。

まずは手っ取り早く、機械語で PCG を定義するプログラムをリスト 2-2 に示す。リスト



2-3 は例によって、BASIC からの使用方法である。リスト 2-2, 2-3 は極めて基本技であるから、あくまでも学習用ということをご心得ておいていただきたい。

リスト 2-2 を説明するに先立って、PCG 定義の原理を説明しておこう。

第 1 章でも示したように、X1 の画面表示は CRTC (CRT コントローラ) が取り仕切っているのである。ここでちょっと脇道にそれて、CRTC がどうやって文字を表示しているのかを、てきとーに考えてみるのである。CRTC はまず画面の左上から、横に走査線 1 本分 (ラスタという) を表示するのである。CRTC は書き始めるに先立って、VRAM を見るのである (ここらへんは、極めていい加減な表現だから注意)。まずは左上の LOCATE 0, 0 の位置だから、I/O 空間の 3000<sub>H</sub> 番地にある VRAM の ASCII コードを見るわけだ。ここでは「A」つまり CHR\$(&H41) が VRAM に書いてあるとしよう。すると CRTC は「おっ! 41<sub>H</sub>だな。それでは、キャラクタジェネレータの 41<sub>H</sub>に対応するところからパターンを持ってきて表示しましょう」とつぶやくのである。ただし、「A」という文字は 8 × 8 ドットだから「上から何段目」ということも指定しなければならない。CRTC はそこんともちゃんと心得ていて、「ASCII コード 41<sub>H</sub>の 0 段目のパターン」というように指定してくるのである。この 8 ドットを表示した後は「VRAM の 3001<sub>H</sub>番地に書かれている ASCII コードの文字の 0 段目のパターン」、……と続くのである。そして、右上の端まで行ったら、同じ ASCII コードに対して段数を増加して 2 本目を左端から表示するのである。

注目して欲しいのは、これらの途中で、「CRTC が CG のパターン (データ) にアクセスしている」ということである。つまり、

### **CRTC はパターンが格納されている ROM や RAM にアクセスしている**

のである。そこで PCG が、ある VRAM にアクセスしているとき、その VRAM に定義の対象になる ASCII コードを用意しておく、CRTC は律儀にその ASCII コードに対応する CG ROM (または RAM) にアクセスする。そのタイミングをねらって、別の回路からデータを送ってやり、パターンを書き込んでしまうのである。これが X1 における PCG 定義の仕組みである。

以上が原理なのだが、よく考えてみると落とし穴があるのだ。どこにあるかというと、PCG が CRTC にアクセスされてデータが読み出されている最中に、横からシャシャリ出てきて、その PCG に読み書き動作をするなど許されないということなのだ。普通のメモリに対して「読むことと書くことを同時に行なう」のは許されないのである (現在はデュアルポート RAM といって、それができる RAM もあるが)。

では、本当のところはどうやっているのかというと、

**CRTC が「表示されない VRAM」にアクセスしているときだけ、他の回路からのデータ R/W を許している**

のである。

そのココロは、VRAM は 3000<sub>H</sub>~37FF<sub>H</sub>=2048 バイトあるのに、画面表示に使われてる VRAM は、普通は、最大で 80×25=2000 バイトで、48 バイトが宙に浮いているということなのである。CRTC がその VRAM に空しいアクセスをしているスキをねらって、デー



タの読み書きを行なうのである。その「表示はされないが、CRTC からアクセスされる VRAM」というのは、

- 1) WIDTH 40: SCREEN, 0 の場合  
I/O の 33E8<sub>H</sub>~33FF<sub>H</sub>までの 24 バイト
- 2) WIDTH 40: SCREEN, 1 の場合  
I/O の 37E8<sub>H</sub>~37FF<sub>H</sub>までの 24 バイト
- 3) WIDTH 80 の場合  
I/O の 37D0<sub>H</sub>~37FF<sub>H</sub>までの 48 バイト

の三つの場合がある。実際はそれぞれに対応するアトリビュート (turbo の場合は、さらに漢字 VRAM も) セットしておく必要がある。

そこでやっとリスト 2-2, 2-3 の説明に入るのであった。

リスト 2-2 機械語による PCG 定義

		.Z80	
		.PHASE 0E000H	
		;	
E000	EB	EX	DE,HL
E001	46	LD	B,(HL) ;HI
E002	23	INC	HL
E003	4E	LD	C,(HL) ;LOW
E004	23	INC	HL
E005	ED 43 E078	LD	(BCWORK),BC
E009	56	LD	D,(HL) ;COUNT
E00A	23	INC	HL
E00B	5E	LD	E,(HL) ;ASCII CODE
E00C	23	INC	HL
E00D	CD E018	CALL	SET0
		;	
E010	46	LD	B,(HL)
E011	23	INC	HL ;HL POINTS PCG DATA
E012	0E 00	LD	C,00H ;BC=1500H,1600H or 1700H
E014	CD E052	CALL	SETPCG
E017	C9	RET	
		;	
E018	E5	SET0:	PUSH HL ;SAVE POINTER
E019	D5		PUSH DE ;SAVE COUNT and ASCII CO
DE			
E01A	01 1FD0	LD	BC,1FD0H ;FOR turbo
E01D	AF	XOR	A
E01E	ED 79	OUT	(C),A
		;	
E020	ED 4B E078	LD	BC,(BCWORK)
E024	21 3800	LD	HL,3800H ;KANJI VRAM(turb
o)			
E027	3E 00	LD	A,00H
E029	CD E048	CALL	SET1
		;	
E02C	ED 4B E078	LD	BC,(BCWORK)
E030	21 2800	LD	HL,2800H ;ATTRIBUTE
E033	D1	POP	DE
E034	D5	PUSH	DE
E035	3E 20	LD	A,20H ;PCG,COLOR=0
E037	CD E048	CALL	SET1
		;	
E03A	ED 4B E078	LD	BC,(BCWORK)
E03E	21 3000	LD	HL,3000H ;VRAM
E041	D1	POP	DE
E042	7B	LD	A,E ;ASCII CODE
E043	CD E048	CALL	SET1
		;	
E046	E1	POP	HL
E047	C9	RET	
		;	
E048	09	SET1:	ADD HL,BC
E049	44		LD B,H
E04A	4D		LD C,L ;BC=HL+BC
E04B	ED 79	SET2:	OUT (C),A



E04D	03	INC	BC	;INC ADDRESS
E04E	15	DEC	D	;DEC COUNTER
E04F	20 FA	JR	NZ,SET2	
E051	C9	RET		
E052	1E 08	SETPCG: LD	E,08H	;COUNTER
E054	D9	EXX		
E055	F3	DI		
E056	01 1A01	LD	BC,1A01H	
E059	ED 78	VDISP0: IN	A,(C)	
E05B	F2 E059	JP	P,VDISP0	
E05E	ED 78	VDISP1: IN	A,(C)	
E060	FA E05E	JP	M,VDISP1	
E063	D9	EXX		
E064	7E	SETP: LD	A,(HL)	;7,GET DATA
E065	ED 79	OUT	(C),A	;12,SET 1 BYTE
E067	23	INC	HL	;6,INC POINTER
E068	03	INC	BC	;6,INC I/O ADDRESS
				;7+12+6+6=31
E069	00	NOP		;4
E06A	23	INC	HL	;6
E06B	2B	DEC	HL	;6,DUMMY
				;4+6+6=16
E06C	3E 0D	LD	A,0DH	;7
E06E	3D	DLY: DEC	A	;4
E06F	C2 E06E	JP	NZ,DLY	;10
E072	1D	DEC	E	;4
E073	C2 E064	JP	NZ,SETP	;10
				;7+(4+10)*13+4+10=203
				;31+16+203=250
E076	FB	EI		
E077	C9	RET		
E078		BCWORK: DS	2	
		END		

リスト 2-2 で最初にやっているのは、パラメータの受け取りである。例によって DE レジスタを利用している。まずは DE レジスタの内容を HL レジスタに移して、BC レジスタに格納している。これは先程説明した「表示されない VRAM」の先頭アドレス(相対値)である。念のために書くが、なぜいちいち受け取っているかというと、そのアドレスは画面モードによって違って来るからである。その次に D レジスタに格納している「COUNT」も画面モードによって変わるのと同じことである。

次に肝心の ASCII コードである。これは以上のパラメータを持って SET0 をコールしている。SET0 では (1FD0<sub>H</sub>) ← 00<sub>H</sub>としているが、これは turbo 用である。turbo ではここが画面管理を行っており、X1 と同じ動作をさせるためには、00<sub>H</sub>を OUT しておく必要がある。ただしこれは念のためであって、X1 モードで起動しておけば、IPL が自動的に 00<sub>H</sub>をセットしておいてくれるから深く悩む必要はない。それは次の漢字 VRAM でも同じで、3800<sub>H</sub>+BC レジスタから、D レジスタ分 00<sub>H</sub>をセットしている。もし turbo でなかった場合は、これは 3000<sub>H</sub>+BC レジスタ番地からの VRAM にセットされることになるが、その場合は直後にまた VRAM へ ASCII コードをセットするので、ただ単にムダになるだけであって、画面が乱れるなどの悪影響はないので安心するよーに。それからアトリビュートも忘れてはいけない。こいつは 20<sub>H</sub> をセットしているから、つまりはノーマルの PCG で、色は 0 (つまり透明) である。ここは第 5 ビットが立っていれば何でもよいので、気分によっては FF<sub>H</sub>などにしてもよいだろう。



SET0 はこれぐらいで、次は御本尊の SETPCG である。まず E010<sub>H</sub>で、あらかじめ B レジスタにパラメータを渡している点に注意。この時点で B レジスタには、15<sub>H</sub>, 16<sub>H</sub>, 17<sub>H</sub>のうちどれかが入り、それによって青、赤、緑のうちどれがセットされるかが決まるのだ。図 2-2 を見て納得していただきたい。

図 2-2 CG, 漢字 ROM, PCG アクセス用 I/O ポート

X1, X1 turbo コンバチアクセスモード		
14* <sub>H</sub>	CG ROMアクセス	IN
15* <sub>H</sub>	PCG BLUEアクセス	IN/OUT
16* <sub>H</sub>	PCG REDアクセス	IN/OUT
17* <sub>H</sub>	PCG GREENアクセス	IN/OUT
X1 turbo 高速アクセスモード		
14*0 <sub>H</sub> ~14*F <sub>H</sub>	CG, 漢字ROMアクセス	IN
15*0 <sub>H</sub> ~15*F <sub>H</sub>	PCG BLUEアクセス	IN/OUT
16*0 <sub>H</sub> ~16*F <sub>H</sub>	PCG REDアクセス	IN/OUT
17*0 <sub>H</sub> ~17*F <sub>H</sub>	PCG GREENアクセス	IN/OUT
チェックポイント		
turboの場合、I/Oポート1FD* <sub>H</sub> の第5ビットが1ならば高速アクセスモード、0ならばコンバチアクセスモード。		

さて、よく見ると分かるように、実はこの方式では、一度に1色分しかセットできないのである。すなわち、正しく PCG をセットするには、このルーチンを3回呼び出さなければならないのだ。この点を心得ておいて欲しいわけである。

では E052<sub>H</sub>からの SETPCG である。まず E レジスタに 08<sub>H</sub>をロードしている。これはとりもなおさず、キャラクタのデータが8段飾りであることに由来する。つまり 08<sub>H</sub>というのはカウンタなのである。その後 E055<sub>H</sub>~E062<sub>H</sub>で何やら行なっているが、これは「垂直帰線期間信号の立ち下がり」(I/O ポート 1A01<sub>H</sub>番地の第7ビット)の検出なのである。「そんなもんを検出して、一体どーすんだ」と思うであろうが、これには「今から CRTC は例の表示されない VRAM へアクセスを開始しま〜す」という意味があるのだ。よってこれが GO サインなのである。ところで、見慣れない条件 JP があるが、これは各自で自習していただきたい。

いよいよ核心中の核心が、E064<sub>H</sub>からである。まず、(BC) へ1バイト OUT している。これで PCG セットが1段分できたことになる。E067<sub>H</sub>までは実にありふれたプログラムである。そして、出ました、ディレイルーチンである。プログラムの右端に並んでいるのは、ステート数(クロック数: X1 は 4MHz だから1クロックが 0.25 μs = 0.00000025 秒)である。ループ1回分が250クロックになるようにしてある。なぜこのようになっていくかというと、これは CRTC の都合によるものなのだ。すなわち「PCG のセットは CRTC が表示されない RAM にアクセスしているときを利用する」という大原則があるので、「CRTC 様が次の段にアクセスなされるとき」をひたすら待たなければならないのである。なぜ250クロックかというと、これは映像信号の水平周期=15.98 kHz からきている



のだ。すなわち、

$$\frac{1}{15.98(\text{kHz})} \doteq 0.0000625 \text{ (秒)}$$
$$= \frac{250 \text{ (クロック)}}{4(\text{MHz})}$$

なのだ。15.98 kHz というのがどこから出てきたかという、資料に書いてあったからなのだ。私は深く追及する気はないので、さっさとリスト 2-3 の説明に入ってしまうのである。

#### リスト 2-3 低速 PCG 定義

```
100 CLEAR &HE000
110 MEM$(&HE000,16)=HEXCHR$("EB 46 23 4E 23 ED 43 78 E0 56 23 5E 23 CD 18 E0")
120 MEM$(&HE010,16)=HEXCHR$("46 23 0E 00 CD 52 E0 C9 E5 D5 01 D0 1F AF ED 79")
130 MEM$(&HE020,16)=HEXCHR$("ED 4B 78 E0 21 00 38 3E 00 CD 48 E0 ED 4B 78 E0")
140 MEM$(&HE030,16)=HEXCHR$("21 00 28 D1 D5 3E 20 CD 48 E0 ED 4B 78 E0 21 00")
150 MEM$(&HE040,16)=HEXCHR$("30 D1 7B CD 48 E0 E1 C9 09 44 4D ED 79 03 15 20")
160 MEM$(&HE050,16)=HEXCHR$("FA C9 1E 08 D9 F3 01 01 1A ED 78 F2 59 E0 ED 78")
170 MEM$(&HE060,16)=HEXCHR$("FA 5E E0 D9 7E ED 79 23 03 00 23 2B 3E 0D 3D C2")
180 MEM$(&HE070,16)=HEXCHR$("6E E0 1D C2 64 E0 FB C9 00 00 00 00 00 00 00 00")
190 DEFUSR0=&HE000
200 '
210 CGEN 1:LOCATE 0,0
220 FOR I=0 TO 255:PRINT #0,CHR$(I);:NEXT:PRINT
230 CGEN 0
240 '
250 TIME=0
260 FOR I=0 TO 255
270   A$=USR0(CHR$(&H7,&HD0,48,I,&H15)+STRING$(8,&H3))
280   A$=USR0(CHR$(&H7,&HD0,48,I,&H16)+STRING$(8,&HC))
290   A$=USR0(CHR$(&H7,&HD0,48,I,&H17)+STRING$(8,&H30))
300 NEXT
310 PRINT TIME
320 TIME=0
330 FOR I=0 TO 255:DEFCHR$(I)=STRING$(24,0):NEXT
340 PRINT TIME
350 '
360 ' A$=USR0(CHR$(&H7,&HD0,48,?,&H??)+STRING$(8,&H??)):WIDTH 80
370 ' A$=USR0(CHR$(&H7,&HE8,24,?,&H??)+STRING$(8,&H??)):WIDTH 40:SCREEN ?,0
380 ' A$=USR0(CHR$(&H7,&HE8,24,?,&H??)+STRING$(8,&H??)):WIDTH 40:SCREEN ?,1
```

まず 210~230 行ですべての PCG を表示しておく。次に 260~300 行ですべての PCG を 3 色縦縞にしてしまう。CHR\$(△) の中のパラメータは、WIDTH 80 のときのものである(他のスクリーンモードのときは 360 行からの注釈文を見ておくれ)。そして、すぐ後では DEFCHR\$文で同じようなことをやっている。結局は速度比較なのである。で、その速度はどーかという、これがなんと同じで、両方とも 256 個の PCG を定義するのに 13 秒前後かかっているのである。考えてみれば、CRT の垂直同期周波数は 60Hz 前後である。つまり、1 秒間に 60 回 PCG をセットするチャンスが来るわけだ。256 個セットするには、1 個当たり青赤緑の 3 回のチャンスが必要だから、256×3=768 回のチャンスが必要である。そいでもって、768÷60=12.8 (秒) かかってしまうのである。

そこで出てくるのが、3 倍速 PCG 定義プログラムであった。

## PCG の高速定義

まずは、3 倍の速さにするのである。この原理は私が考えたものではないということ



最初に断っておく。

リスト 2-4, 2-5 を見ていただきたい。注意してみると分かるように、リスト 2-4 の前半はリスト 2-2 とほとんど同じである。違ってくるのは、E04E<sub>H</sub>の SETPCG からである。まずは ORDER というサブルーチンを呼んでいる。ここで何をやっているかという、青、赤、緑と 8 バイトずつのブロックになっているデータを、青の 0 段目、赤の 0 段目、緑の 0 段目、……という順序に並べ変えて、WORK (E0AC<sub>H</sub>) から始まる 24 バイトに格納しているのである。E08D<sub>H</sub>~E0A3<sub>H</sub>あたりの 2 重ループの作り方が実に微妙なのである。特に注意すべきことは「LDI」を実行して BC レジスタが 0 になったかどうかの判断を、「JP PO, ~」(パリティ・オッド)で判定している点である。「LDI」はゼロフラグを変えないのである。

リスト 2-4 3 倍速 PCG 定義の機械語部分

		.Z80	
		.PHASE	0E000H
0016		;	
0017		BLUE EQU	15H+1
0018		RED EQU	16H+1
		GREEN EQU	17H+1
		;	
E000	EB	EX	DE,HL
E001	46	LD	B,(HL) ;HI
E002	23	INC	HL
E003	4E	LD	C,(HL) ;LOW
E004	23	INC	HL
E005	ED 43 E0AA	LD	(BCWORK),BC
E009	56	LD	D,(HL) ;COUNT
E00A	23	INC	HL
E00B	5E	LD	E,(HL) ;ASCII CODE
E00C	23	INC	HL
E00D	CD E014	CALL	SET0
		;	
		;	HL POINTS PCG DATA
E010	CD E04E	CALL	SETPCG
E013	C9	RET	
		;	
E014	E5	SET0:	PUSH HL ;SAVE POINTER
E015	D5		PUSH DE ;SAVE COUNT & ASCII CODE
E016	01 1FD0		LD BC,1FD0H ;FOR turbo
E019	AF		XOR A
E01A	ED 79		OUT (C),A
		;	
E01C	ED 4B E0AA		LD BC,(BCWORK)
E020	21 3800		LQ HL,3800H ;KANJI VRAM:turb
		;	
E023	3E 00		LD A,00H
E025	CD E044		CALL SET1
		;	
E028	ED 4B E0AA		LD BC,(BCWORK)
E02C	21 2800		LD HL,2800H ;ATTRIBUTE
E02F	D1		POP DE
E030	D5		PUSH DE
E031	3E 20		LD A,20H ;PCG,COLOR=0
E033	CD E044		CALL SET1
		;	
E036	ED 4B E0AA		LD BC,(BCWORK)
E03A	21 3000		LD HL,3000H ;VRAM
E03D	D1		POP DE
E03E	7B		LD A,E ;ASCII CODE
E03F	CD E044		CALL SET1
		;	
E042	E1		POP HL
E043	C9		RET
		;	
E044	09	SET1:	ADD HL,BC
E045	44		LD B,H
E046	4D		LD C,L ;BC=HL+BC
E047	ED 79	SET2:	OUT (C),A
E049	03		INC BC ;INC ADDRESS
E04A	15		DEC D ;DEC COUNTER



E04B	20 FA	JR	NZ,SET2
E04D	C9	RET	
		; Acc=ASCII	
E04E	CD E086	SETPCG: CALL	ORDER ;CHANGE ORDER
		;	
E051	06 16	LD	B,BLUE
E053	0E 00	LD	C,00H
E055	16 17	LD	D,RED ;RED
E057	1E 18	LD	E,GREEN ;GREEN
E059	3E 08	LD	A,08H ;COUNTER
E05B	08	EX	AF,AF'
E05C	D9	EXX	
		;	
E05D	F3	DI	
E05E	01 1A01	LD	BC,1A01H
E061	ED 78	VDSP0: IN	A,(C)
E063	F2 E061	JP	P,VDSP0
E066	ED 78	VDSP1: IN	A,(C)
E068	FA E066	JP	M,VDSP1
		;	
E06B	D9	EXX	
E06C	08	EX	AF,AF' ;A'=COUNTER
		;	
E06D	ED A3	SETP: OUTI	;16,OUT BLUE
E06F	42	LD	B,D ;4
E070	ED A3	OUTI	;16,OUT RED
E072	43	LD	B,E ;4
E073	ED A3	OUTI	;16,OUT GREEN
		;16+4+16+4+16=56	
		;	
E075	06 16	LD	B,BLUE ;7
		;7=7	
		;	
E077	08	EX	AF,AF' ;4
E078	3E 0B	LD	A,0BH ;7
E07A	3D	DLY: DEC	A ;4
E07B	C2 E07A	JP	NZ,DLY ;10
E07E	08	EX	AF,AF' ;4
		;4+7+(4+10)*11+4=169	
		;	
E07F	0C	INC	C ;4
E080	3D	DEC	A ;4
E081	C2 E06D	JP	NZ,SETP ;10
		;4+4+10=18	
		;56+7+169+18=250	
E084	FB	EI	
E085	C9	RET	
		;	
		;CHANGE ORDER	
		;B0,B1,...,B7,R0,R1,...,R7,G0,G1,...,G7	
		;---->B0,R0,G0,B1,R1,G1,...,B7,R7,G7	
		;	
E086	D9	ORDER: EXX	
E087	06 08	LD	B,8 ;COUNTER
E089	D9	EXX	
E08A	11 E0AC	LD	DE,WORK ;BUFFER
E08D	E5	ORDER0: PUSH	HL
E08E	01 0003	LD	BC,3
E091	ED A0	ORDER1: LDI	
		;(DE)<-(HL),INC DE,INC HL,DEC BC	
E093	E2 E09E	JP	PO,ORDER2 ;3 BYTES MOVED?
E096	79	LD	A,C ;SAVE C
E097	0E 07	LD	C,7
E099	09	ADD	HL,BC ;HL=HL+7
E09A	4F	LD	C,A ;BACK C
E09B	C3 E091	JP	ORDER1
		;	
E09E	E1	ORDER2: POP	HL
E09F	23	INC	HL
E0A0	D9	EXX	
E0A1	05	DEC	B ;DEC COUNTER
E0A2	D9	EXX	
E0A3	C2 E08D	JP	NZ,ORDER0
		;	
E0A6	21 E0AC	LD	HL,WORK ;NEW PCG DATA
E0A9	C9	RET	
		;	
E0AA		BCWORK: DS	2
E0AC		WORK: DS	24
		END	



## リスト 2-5 3 倍速 PCG 定義

```

100 CLEAR &HE000
110 MEM$(&HE000,16)=HEXCHR$("EB 46 23 4E 23 ED 43 AA E0 56 23 5E 23 CD 14 E0")
120 MEM$(&HE010,16)=HEXCHR$("CD 4E E0 C9 E5 D5 01 D0 1F AF ED 79 ED 4B AA E0")
130 MEM$(&HE020,16)=HEXCHR$("21 00 38 3E 00 CD 44 E0 ED 4B AA E0 21 00 28 D1")
140 MEM$(&HE030,16)=HEXCHR$("D5 3E 20 CD 44 E0 ED 4B AA E0 21 00 30 D1 7B CD")
150 MEM$(&HE040,16)=HEXCHR$("44 E0 E1 C9 09 44 4D ED 79 03 15 20 FA C9 CD 86")
160 MEM$(&HE050,16)=HEXCHR$("E0 06 16 0E 00 16 17 1E 18 3E 08 08 D9 F3 01 01")
170 MEM$(&HE060,16)=HEXCHR$("1A ED 78 F2 61 E0 ED 78 FA 66 E0 D9 08 ED A3 42")
180 MEM$(&HE070,16)=HEXCHR$("ED A3 43 ED A3 06 16 08 3E 0B 3D C2 7A E0 08 0C")
190 MEM$(&HE080,16)=HEXCHR$("3D C2 6D E0 FB C9 D9 06 08 D9 11 AC E0 E5 01 03")
200 MEM$(&HE090,16)=HEXCHR$("00 ED A0 E2 9E E0 79 0E 07 09 4F C3 91 E0 E1 23")
210 MEM$(&HE0A0,16)=HEXCHR$("D9 05 D9 C2 8D E0 21 AC E0 C9 00 00 00 00 00 00")
220 DEFUSR0=&HE000
230 '
240 CGEN 1:LOCATE 0,0
250 FORI=0TO255:PRINT #0,CHR$(I);:NEXT:PRINT
260 CGEN 0
270 '
280 TIME=0
290 FORI=0TO255
300   A$=USR0(CHR$(&H7,&HD0,48,I)+STRING$(24,1))
310 NEXT
320 PRINT TIME

```

このようにして並べ変えたデータを E06D<sub>H</sub> から OUTI を使って、走査線 1 本当たり、青、赤、緑の 3 バイトずつ PCG にセットしているわけである。ここでの注意点は、OUTI 命令のひねくれ方である。実にムツとすることにより、OUTI は、B レジスタを一つ減少させてから、OUT 動作を行なうのである。よって、1500<sub>H</sub> 番地に OUT したいならば、BC を 1600<sub>H</sub> にしておかなければならないのである。むむむーん。あと特筆すべきことは、その 3 バイトを OUT する期間である。どうやら 60 クロックぐらいの間に済まなければならないよーである。しかし、私はその根拠は計算できない。ま、56 クロックで収まっているのだからよしとしてしまうのである。

この 3 倍速 PCG 定義を使うと、256 個の PCG 定義が 4 秒強で済むことになる。しかしこの世にその筋として生を受けたからには、「もっとオイシイ手はないかかかっ!」と考えなければならん。そして思い付いたのが、24 倍速 PCG 定義である。一体どーやってやるのかというと、原理は単純で、

**PCG 定義で大事な「表示されない VRAM」の数(行数)を増やしてしまおう。**

という、実にその筋的な発想なのである。第 1 章で CRTC をやったが、CRTC の第 6 レジスタ (R6) は「垂直表示文字数」であった。普段はここは 25 なのだが、7 減らして 18 にしてしまうと、どーだ「表示されない VRAM」がどっと増えただろーが。7 行増えたのだから、3 倍速×8 倍速で 24 倍速である。機械語部分がリスト 2-6、BASIC から使っているのがリスト 2-7 である。リスト 2-6 でやっているのは、CRTC をいじっているほかは、リスト 2-4 の 3 倍速 PCG 定義と大差ない。そこでさっさとリスト 2-7 の説明に行ってしまうのである。

## リスト 2-6 24 倍速 PCG 定義の機械語部分

		.280	
		.PHASE	0E000H
	;		
0016	BLUE	EQU	15H+1
0017	RED	EQU	16H+1
0018	GREEN	EQU	17H+1
	;		



E000	01 1800	LD	BC,1800H	;SET CRTC
E003	3E 06	LD	A,6	
E005	ED 79	OUT	(C),A	
E007	03	INC	BC	
E008	3E 12	LD	A,18	;DISP JUST 18 LINES
E00A	ED 79	OUT	(C),A	
;				
E00C	01 1FD0	START: LD	BC,1FD0H	;FOR turbo
E00F	AF	XOR	A	
E010	ED 79	OUT	(C),A	
;				
E012	01 3DA0	LD	BC,3800H+5A0H	;KANJI VRAM
E015	24 0260	LD	HL,260H	;COUNTER
E018	16 00	LD	D,00H	
E01A	CD E070	CALL	SETIO	
;				
E01D	01 25A0	LD	BC,2000H+5A0H	;ATTRIBUTE
E020	21 0260	LD	HL,260H	;COUNTER
E023	16 20	LD	D,20H	
E025	CD E070	CALL	SETIO	
;				
E028	21 E200	LD	HL,0E200H	;DATA AREA
E02B	22 E0AF	LD	(HLWORK),HL	;SAVE ADDRESS
E02E	AF	XOR	A	;TOP OF ASCII CODE
E02F	F5	LOOP: PUSH	AF	
E030	CD E056	CALL	SET8	
E033	2A E0AF	LD	HL,(HLWORK)	;DATA ADDRESS
E036	CD E07A	CALL	SETPCG	
E039	2A E0AF	LD	HL,(HLWORK)	
E03C	01 00C0	LD	BC,24*8	
E03F	09	ADD	HL,BC	;NEW ADDRESS
E040	22 E0AF	LD	(HLWORK),HL	
E043	F1	POP	AF	
E044	C6 08	ADD	A,8	;CHECK COUNTER
E046	C2 E02F	JP	NZ,LOOP	
;				
E049	01 1800	LD	BC,1800H	
E04C	3E 06	LD	A,6	
E04E	ED 79	OUT	(C),A	
E050	03	INC	BC	
E051	3E 19	LD	A,25	;CRTC WO MODOSU
E053	ED 79	OUT	(C),A	
E055	C9	RET		
;				
E056	01 35A0	SET8: LD	BC,3000H+5A0H	
E059	57	LD	D,A	;ASCII CODE
E05A	1E 08	LD	E,8	;COUNTER
E05C	C5	SET80: PUSH	BC	
E05D	21 0030	LD	HL,48	
E060	CD E070	CALL	SETIO	
E063	C1	POP	BC	
E064	21 0050	LD	HL,80	
E067	09	ADD	HL,BC	
E068	44	LD	B,H	
E069	4D	LD	C,L	;BC=NEXT ADDRESS
E06A	14	INC	D	;INC ASCII CODE
E06B	1D	DEC	E	;DEC COUNTER
E06C	C2 E05C	JP	NZ,SET80	
;				
E06F	C9	RET		
;				
E070	ED 51	SETIO: OUT	(C),D	
E072	03	INC	BC	
E073	2B	DEC	HL	
E074	7C	LD	A,H	
E075	B5	OR	L	
E076	C2 E070	JP	NZ,SETIO	
E079	C9	RET		
;				
E07A	06 16	;Acc=ASCII		
E07C	0E 00	SETPCG: LD	B,BLUE	
E07E	16 17	LD	C,00H	
E080	1E 18	LD	D,RED	;RED
E082	3E 40	LD	E,GREEN	;GREEN
E084	08	LD	A,08H*8	;COUNTER,8 CHARS!
E085	D9	EX	AF,AF'	
;				
E086	F3	DI		
E087	01 1A01	LD	BC,1A01H	
E08A	ED 78	VDSP0: IN	A,(C)	



E08C	F2 E08A	JP	P,VDSP0	
E08F	ED 78	VDSP1: IN	A,(C)	
E091	FA E08F	JP	M,VDSP1	
		;		
E094	D9	EXX		
E095	08	EX	AF,AF'	;A'=COUNTER
		;		
E096	ED A3	SETP: OUTI		;16,OUT BLUE
E098	42	LD	B,D	;4
E099	ED A3	OUTI		;16,OUT RED
E09B	43	LD	B,E	;4
E09C	ED A3	OUTI		;16,OUT GREEN
				;16+4+16+4+16=56
		;		
E09E	06 16	LD	B,BLUE	;7
				;7=7
		;		
E0A0	08	EX	AF,AF'	;4
E0A1	3E 0B	LD	A,0BH	;7
E0A3	3D	DLY: DEC	A	;4
E0A4	C2 E0A3	JP	NZ,DLY	;10
E0A7	08	EX	AF,AF'	;4
				;4+7+(4+10)*11+4=169
		;		
E0A8	0C	INC	C	;4
E0A9	3D	DEC	A	;4
E0AA	C2 E096	JP	NZ,SETP	;10
				;4+4+10=18
				;56+7+169+18=250
E0AD	FB	EI		
E0AE	C9	RET		
		;		
E0AF		HLWORK: DS	2	
		END		

リスト 2-7 24 倍速 PCG 定義

100	CLEAR &HE000
110	MEM\$(&HE000,16)=HEXCHR\$("01 00 18 3E 06 ED 79 03 3E 12 ED 79 01 D0 1F AF")
120	MEM\$(&HE010,16)=HEXCHR\$("ED 79 01 A0 3D 21 60 02 16 00 CD 70 E0 01 A0 25")
130	MEM\$(&HE020,16)=HEXCHR\$("21 60 02 16 20 CD 70 E0 21 00 E2 22 AF E0 AF F5")
140	MEM\$(&HE030,16)=HEXCHR\$("CD 56 E0 2A AF E0 CD 7A E0 2A AF E0 01 C0 00 09")
150	MEM\$(&HE040,16)=HEXCHR\$("22 AF E0 F1 C6 08 C2 2F E0 01 00 18 3E 06 ED 79")
160	MEM\$(&HE050,16)=HEXCHR\$("03 3E 19 ED 79 C9 01 A0 35 57 1E 08 C5 21 30 00")
170	MEM\$(&HE060,16)=HEXCHR\$("CD 70 E0 C1 21 50 00 09 44 4D 14 1D C2 5C E0 C9")
180	MEM\$(&HE070,16)=HEXCHR\$("ED 51 03 2B 7C B5 C2 70 E0 C9 06 16 0E 00 16 17")
190	MEM\$(&HE080,16)=HEXCHR\$("1E 18 3E 40 08 D9 F3 01 01 1A ED 78 F2 8A E0 ED")
200	MEM\$(&HE090,16)=HEXCHR\$("78 FA 8F E0 D9 08 ED A3 42 ED A3 43 ED A3 06 16")
210	MEM\$(&HE0A0,16)=HEXCHR\$("08 3E 0B 3D C2 A3 E0 08 0C 3D C2 96 E0 FB C9 00")
220	'
230	WIDTH 80
240	CGEN 1
250	FOR I=0 TO 255:PRINT #0 CHR\$(I);:NEXT:PRINT
260	CGEN 0
270	FOR I=0 TO 255
280	MEM\$(&HE200+I*24,24)=STRING\$(24,I)
290	NEXT
300	TIME=0
310	FOR I=0 TO 10:CALL &HE000:NEXT
320	PRINT TIME

リスト 2-7 では最初に WIDTH を 80 にしている。そーなのである。リスト 2-6 は簡単にするために、WIDTH 80 でしか動かないよーになっているのだ。しかも PCG 用のデータは、あらかじめ青、赤、緑、青、赤、緑、……というように並べ換えられて、E200<sub>H</sub>から 6144 (1800<sub>H</sub>) バイトに格納されていることになっている。270~290 行で、そのダミーのデータを作っているわけである。実際の PCG 定義は「CALL &HE000」一発で OK である。速すぎるので 10 回実行している。結果は、256 個の定義にかかる時間は約 0.6 秒と出た。さて、理論的には全画面を表示されない VRAM (?) にして 26×3=78 倍速なんていうのができそうだが、ちょっと無理のようである。実は 16×3=48 倍速までやってみた



のだが、ソフトウェアでタイミングをとる悲しさで、所々で PCG の定義に失敗してしまうのだ。それで安全のために 24 倍速でとどめておいたのである(その筋な某ユーザーは、もう一工夫して、うまくソフトウェアでタイミングをとり、48 倍速定義を成功させたそうである。ももも)。

## CGの読み出し

話は変わって、BASIC の CGPAT\$関数に対応する部分である。リスト 2-8, 2-9 に ROM CG を読み出すプログラムを示す。リスト 2-2~7 を参考にすれば、PCG から読み出すことなどは簡単であろうと、勝手に判断してしまうのである。達者で暮らしていただきたい、と言いつつ何の説明もせずに次に進んでしまうのである。

リスト 2-8 ROM CG 読み出しの機械語部分

			.Z80	
			.PHASE	0E000H
		;		
E000	EB		EX	DE,HL
E001	46		LD	B,(HL) ;HI
E002	23		INC	HL
E003	4E		LD	C,(HL) ;LOW
E004	23		INC	HL
E005	ED 43 E077		LD	(BCWORK),BC
E009	56		LD	D,(HL) ;COUNT
E00A	23		INC	HL
E00B	5E		LD	E,(HL) ;ASCII CODE
E00C	23		INC	HL
E00D	CD E017		CALL	SET0
		;		
E010	01 1400		LD	BC,1400H
E013	CD E051		CALL	READCG
E016	C9		RET	
		;		
E017	E5	SET0:	PUSH	HL ;SAVE POINTER
E018	D5		PUSH	DE ;SAVE COUNT and ASCII CO
DE				
E019	01 1FD0		LD	BC,1FD0H ;FOR turbo
E01C	AF		XOR	A
E01D	ED 79		OUT	(C),A
		;		
E01F	ED 4B E077		LD	BC,(BCWORK)
E023	21 3800		LD	HL,3800H ;KANJI VRAM(turb
o)				
E026	3E 00		LD	A,00H
E028	CD E047		CALL	SET1
		;		
E02B	ED 4B E077		LD	BC,(BCWORK)
E02F	21 2800		LD	HL,2800H ;ATTRIBUTE
E032	D1		POP	DE
E033	D5		PUSH	DE
E034	3E 00		LD	A,00H ;ROM CG,COLOR=0
E036	CD E047		CALL	SET1
		;		
E039	ED 4B E077		LD	BC,(BCWORK)
E03D	21 3000		LD	HL,3000H ;VRAM
E040	D1		POP	DE
E041	7B		LD	A,E ;ASCII CODE
E042	CD E047		CALL	SET1
		;		
E045	E1		POP	HL
E046	C9		RET	
		;		
E047	09	SET1:	ADD	HL,BC
E048	44		LD	B,H
E049	4D		LD	C,L ;BC=HL+BC
E04A	ED 79	SET2:	OUT	(C),A
E04C	03		INC	BC ;INC ADDRESS
E04D	15		DEC	D ;DEC COUNTER



E04E	20 FA	JR	NZ,SET2
E050	C9	RET	
E051	1E 08	;	
E053	D9	READCG: LD	E,08H ;COUNTER
E054	F3	EXX	
E055	01 1A01	;	
E058	ED 78	DI	
E05A	F2 E058	LD	BC,1A01H
E05D	ED 78	VDISP0: IN	A,(C)
E05F	FA E05D	JP	P,VDISP0
E062	D9	VDISP1: IN	A,(C)
E063	ED 78	JP	M,VDISP1
E065	77	;	
E066	23	EXX	
E067	03	SETP: IN	A,(C) ;12,READ 1 BYTE
E068	00	LD	(HL),A ;7,STORE DATA
E069	23	INC	HL ;6,INC POINTER
E06A	2B	INC	BC ;6,INC I/O ADDRESS
E06B	3E 0D	;	
E06D	3D	;	
E06E	C2 E06D	;	
E071	1D	;	
E072	C2 E063	;	
E075	FB	;	
E076	C9	;	
E077		;	
		BCWORK: DS	2
		END	

リスト 2-9 ROM CG 読み出し

100	CLEAR &HE000
110	MEM\$(&HE000,16)=HEXCHR\$("EB 46 23 4E 23 ED 43 77 E0 56 23 5E 23 CD 17 E0")
120	MEM\$(&HE010,16)=HEXCHR\$("01 00 14 CD 51 E0 C9 E5 D5 01 D0 1F AF ED 79 ED")
130	MEM\$(&HE020,16)=HEXCHR\$("4B 77 E0 21 00 38 3E 00 CD 47 E0 ED 4B 77 E0 21")
140	MEM\$(&HE030,16)=HEXCHR\$("00 28 D1 D5 3E 00 CD 47 E0 ED 4B 77 E0 21 00 30")
150	MEM\$(&HE040,16)=HEXCHR\$("D1 7B CD 47 E0 E1 C9 09 44 4D ED 79 03 15 20 FA")
160	MEM\$(&HE050,16)=HEXCHR\$("C9 1E 08 D9 F3 01 01 1A ED 78 F2 58 E0 ED 78 FA")
170	MEM\$(&HE060,16)=HEXCHR\$("5D E0 D9 ED 78 77 23 03 00 23 2B 3E 0D 3D C2 6D")
180	MEM\$(&HE070,16)=HEXCHR\$("E0 1D C2 63 E0 FB C9 00 00 00 00 00 00 00 00")
190	DEFUSR0=&HE000
200	'
210	WIDTH80
220	C=0
230	A\$=USR0(CHR\$(&H7,&HD0,48,C)+STRING\$(8,&H30))
240	A\$=RIGHT\$(A\$,8)
250	PRINT #0 A\$
260	PRINT #0 LEFT\$(CGPAT\$(C),8)

## turboにおけるPCG/CG

さて、話は変わって turbo である。ここまでの話からも分かるように、X1 の PCG は強力無比と言えるのだが、PCG 定義、読み出しの速度と手間は実にうっとーしいのである。伝え聞くところによると、まともにインターフェイスなどを作ったらコストがかかりすぎるので、あのようになったということだが、そのウワサの真偽はともかくとして、うっとーしいということだけは真実なのである。シャープもその点については正しく理解していたらしく、turbo では「高速アクセスモード」という機能が付けられている。これ



は、CG アクセスの下準備を VRAM, 漢字 VRAM, アトリビュートに各 1 バイト, 計 3 バイトを書き込むだけで済むようにしたうえに, 垂直帰線期間だけでなく, 水平帰線期間にもアクセス可能にしたものである。つまり, 走査線 1 本ごとにアクセスできるわけである。しかもうれしいことにソフトウェアでタイミングを取る必要もないのである。どーしてかというと, turbo がハード的に, 勝手にタイミングを取ってくれるのだ。すなわち, PCG にデータをセットしようとして, CPU が OUT 命令を実行したとすると, その瞬間にウェイトがかけられてタイミングが取れるのである。ウェイトとは平たく言えば「金縛り」みたいなもので, CPU は気絶させられるわけである。で, 周りの回路が準備 OK になったところで CPU に「活」を入れる。CPU は何も気付かずに OUT 命令を完了して, また 4MHz のステップを踏み続けるのである。

turbo ではこの高速アクセスモードにより, 256 個の PCG を, 低解像度モード (200 ライン) のとき約 0.5 秒, 高解像度モード (400 ライン) のとき約 0.27 秒に短縮できたのである。さもなくば SYMBOL 文などは大ひんしゆくであったろう。めでたしめでたし。

まずは実物を見ていただきたい。リスト 2-10, 2-11 である。リスト 2-10 はこの章のア

リスト 2-10 turbo 用 CG R/W プログラムの機械語部分

			.Z80	
			.PHASE	0E000H
		;		
E000	EB		EX	DE, HL
E001	01 20D0		LD	BC, 1FD0H+100H
E004	ED A3		OUTI	;DATA TO (1FD0H)
		;		
E006	01 40FF		LD	BC, 3FFFH+100H
E009	ED A3		OUTI	;KANJI VRAM
		;		
E00B	01 28FF		LD	BC, 27FFH+100H
E00E	ED A3		OUTI	;ATTRIBUTE
		;		
E010	01 38FF		LD	BC, 37FFH+100H
E013	ED A3		OUTI	;VRAM
		;		
E015	56		LD	D, (HL) ;COUNT
E016	23		INC	HL
E017	5E		LD	E, (HL) ;STEP
E018	23		INC	HL
E019	46		LD	B, (HL) ;14H, 15H, 16H or 17H
E01A	23		INC	HL
E01B	0E 00		LD	C, 00H
E01D	7E		LD	A, (HL)
E01E	23		INC	HL ;HL POINTS DATA AREA
		;		
E01F	B7		OR	A ;A=0 THEN SETPCG ELSE RE
AD				
E020	C2 E02E		JP	NZ, READ
		;		
E023	04	WRITE:	INC	B
E024	ED A3		OUTI	
E026	79		LD	A, C
E027	83		ADD	A, E
E028	4F		LD	C, A ;C=C+STEP
E029	15		DEC	D
E02A	C2 E023		JP	NZ, WRITE
E02D	C9		RET	
		;		
E02E	ED A2	READ:	INI	
E030	04		INC	B
E031	79		LD	A, C
E032	83		ADD	A, E
E033	4F		LD	C, A ;C=C+STEP
E034	15		DEC	D



E035	C2 E02E	JP	NZ, READ
E038	C9	RET	
		END	

リスト 2-11 turbo 用 CG R/W プログラム

```

100 CLEAR &HE000
110 MEM$(&HE000,16)=HEXCHR$("EB 01 D0 20 ED A3 01 FF 40 ED A3 01 FF 28 ED A3")
120 MEM$(&HE010,16)=HEXCHR$("01 FF 38 ED A3 56 23 5E 23 46 23 0E 00 7E 23 B7")
130 MEM$(&HE020,16)=HEXCHR$("C2 2E E0 04 ED A3 79 83 4F 15 C2 23 E0 C9 ED A2")
140 MEM$(&HE030,16)=HEXCHR$("04 79 83 4F 15 C2 2E E0 C9 00 00 00 00 00 00")
150 DEFUSR0=&HE000
160 '
170 WIDTH 40
180 '1 2 3 4 5 6 7 8
190 '1FD0,KANJI,ATT,VRAM,COUNT(16/8),STEP(2/1),I/O PORT HIGH,R/W
200 '
210 ' 8 * 8 の ROM CG を読む (アルファベット)
220 A$=USR0(CHR$(&H23,0,0,66,8,2,&H14,1)+STRING$(48,0)):L=8:GOSUB"DUMP"
230 ' 8 * 16 の ROM CG を読む (アルファベット)
240 A$=USR0(CHR$(&H63,0,0,66,16,1,&H14,1)+STRING$(48,0)):L=16:GOSUB"DUMP"
250 ' 8 * 16 の ROM CG を読む (漢字:『筋』の左側)
260 A$=USR0(CHR$(&H63,&H86,0,&HDA,16,1,&H14,1)+STRING$(48,0)):L=16:GOSUB"DUMP"
270 SUJI$=A$
280 '
290 GOSUB"ALL"
300 '
310 ' PCG をノーマルモードでセット
320 A$=USR0(CHR$(&H23,0,&H20,0,8,2,&H15,0)+STRING$(8,&H3))
330 A$=USR0(CHR$(&H23,0,&H20,0,8,2,&H16,0)+STRING$(8,&HC))
340 A$=USR0(CHR$(&H23,0,&H20,0,8,2,&H17,0)+STRING$(8,&H30))
350 '
360 ' PCG を外字モードでセット
370 A$=USR0(CHR$(&H23,&H10,&H20,4,16,1,&H15,0)+SUJI$)
380 A$=USR0(CHR$(&H23,&H10,&H20,4,16,1,&H16,0)+SUJI$)
390 A$=USR0(CHR$(&H23,&H10,&H20,4,16,1,&H17,0)+SUJI$)
400 '
410 OUT &H31A4,&H4
420 OUT &H39A4,&H10
430 OUT &H21A4,&H27 : ' 左筋を表示
440 END
450 '
460 LABEL"DUMP"
470 A$=MID$(A$,9,L)
480 FOR I=1 TO L
490 B=ASC(MID$(A$,I,1)):B$=RIGHT$("00000000"+BIN$(B),8):PRINTB$
500 NEXT:PRINT
510 Q$=INKEY$(1)
520 RETURN
530 '
540 LABEL"ALL"
550 CLS:KMODE 0
560 CGEN 1:LOCATE 0,0
570 FOR I=0 TO 255:PRINT #0,CHR$(I);:NEXT
580 CGEN 0:KMODE 1
590 RETURN

```

センブラで書かれたプログラムの中では一番短いものだが、なんと turbo の PCG, ROM CG, 漢字 ROM の読み書きすべてに対応してしまっているのだ(多少はナニだけどね)。リスト 2-10 の中のループの E028<sub>H</sub>番地と E033<sub>H</sub>番地で C レジスタを変化させているが、これはちゃんと意味があってやっている。使い方はリスト 2-11 の 180, 190 行を見ていただきたい。パラメータは順に、

- 1FD0<sub>H</sub>へのデータ
- 漢字 VRAM へのデータ
- アトリビュートへのデータ
- VRAM へのデータ



- COUNT (8×8を読むか8×16を読むか)
- STEP (8×8なら2, 8×16なら1)
- I/O ポートの上位アドレス (14<sub>H</sub>~17<sub>H</sub>)
- R/W のフラグ (読む=1/書く=0)

となっている。

210 行からは、まず8×8のROM CGの読み出しである。ASCIIコードは66="B"である。第1パラメータが23<sub>H</sub>で、図2-3を見るとビット6=0, ビット5=1である。すなわち8ラスタCGアクセスで、高速アクセスである。おっと、ビット5の説明では「PCG」となっている。愛敬愛敬。さて、第6パラメータが2になっているが、なぜそのようなになっているかという、8×8の「B」のパターンは、表示されるときは

7C<sub>H</sub>, 42<sub>H</sub>, ……

となっているのだが、読み出すと

7C<sub>H</sub>, 7C<sub>H</sub>, 42<sub>H</sub>, 42<sub>H</sub>, ……

と2度ずつ繰り返してしまうのだ。だから「BC=BC+2」としているのだ。230 行からはビット6=1となり、8×16のCGに対するアクセスである。このビット6により、アクセスするROMが変わるのである(表示されるCGは変わらない)。250 行からの漢字ROMの読み出しも同じようなものである。

次に310 行から2度PCGをセットしている。turboでのPCGには、8×8として扱われるノーマルモードと、8×16として扱われる外字モードの二つのモードがある。外字モードの場合は偶数番目と、その次の奇数番目のPCG 2個が1個の8×16の文字として扱われる(たとえば0と1, 2と3などの組)。よって、16×16の漢字は4個のPCGを合体させて作ることができるわけである。256÷4=64で、turboは64個の漢字の外字を持てる

図2-3 1FD0<sub>H</sub>=画面管理ポートの意味

データ内容	コントロール
ビット 0	0=低解像度モニタ(200ライン) 1=高解像度モニタ(400ライン)      モニタ切り換え
ビット 1	0=1本ラスタ/ドット 1=2本ラスタ/ドット
ビット 2	0=ノーマル (8ラスタ/CHAR)(25行, 20行) 1=漢字 (16ラスタ/CHAR)(12行, 10行)
ビット 3	0=バンク0表示 1=バンク1表示
ビット 4	0=バンク0アクセス 1=バンク1アクセス
ビット 5	0=PCGコンバチアクセス 1=PCG高速アクセス
ビット 6	0=8ラスタCGアクセス 1=16ラスタCGアクセス
ビット 7	0=アンダーラインなし 1=アンダーラインあり



ことになる。410 行からは「筋」の左半分を表示している。漢字 VRAM のビット 4 が立っていて、外字モードになっている点に注意。これにより PCG の 4 番と 5 番が上下になって同時に表示されるわけである。

以上が PCG なわけであった。X1 では PCG を使うことにより ALL BASIC でも、まともに遊べるゲームができてしまうのである。マシン語にすればもっと遊べるのである。というわけで、PCG はゲーム街道をお通りなのであった。







第

3

章

漢 字



漢字名野出亜留



### ■ 第3章

## 漢字名野出亜留 ●●●●●●●●●●

第2章ではPCGをやったのである。となれば、この章では漢字ということになるわけだ。

### 漢字コードの基本

大体の人は知っているように、漢字にも ASCII コードなどと同じように、「漢字コード」なるものがあり、整理整頓に一役買っているのである。そいでもって、この漢字コードというやつは実に食わせものなのだ。まず、恐ろしいことに、この漢字コードには3種類が群雄割拠しているのである。

#### ① JIS 漢字コード

#### ② 区点コード

#### ③ シフト JIS 漢字コード

である。これらは三つとも「2 バイトコード」、つまり、2 バイトで漢字1文字を表現するコードである。

#### ① まず、JIS 漢字コードの場合は、

上位バイト (1 バイト目) は  $21_{\text{H}} \sim 7\text{E}_{\text{H}}$

下位バイト (2 バイト目) も  $21_{\text{H}} \sim 7\text{E}_{\text{H}}$

となっている。ちなみに「鯖」の JIS コードは  $3\text{B}2\text{A}_{\text{H}}$  である。

さて、それぞれのバイトで、 $21_{\text{H}} \sim 7\text{E}_{\text{H}}$ しか使っていないことから分かるように、このコードには「隙間」ができています。たとえば、 $0000_{\text{H}} \sim 2120_{\text{H}}$ が空いてるし、 $217\text{F}_{\text{H}} \sim 2220_{\text{H}}$ なども空いているのである。なんでこうなっているかというと、どうやらコントロールコードを入れたくなかったからのようである。つまり、ASCII コードでは  $00_{\text{H}} \sim 20_{\text{H}}$ は ( $20_{\text{H}}$ のスペースは違うけど) コントロールコードとして特殊な機能を割り当てられているから、避けたようなのである。また、それ以外にも空きがある。たとえば、 $2577_{\text{H}} \sim 257\text{E}_{\text{H}}$ などである。この部分はカタカナとギリシャ文字の隙間なのである。

そいでもって、JIS 漢字コードは第1水準と第2水準に分かれるわけであるが、

第1水準 =  $2121_{\text{H}} \sim 4\text{F}53_{\text{H}}$

第2水準 =  $5020_{\text{H}} \sim 7424_{\text{H}}$

となっている。第1水準の  $4\text{F}54_{\text{H}} \sim 4\text{F}7\text{E}_{\text{H}}$ と、第2水準の  $7425_{\text{H}} \sim 7\text{E}7\text{E}_{\text{H}}$ は未定義 (空き) である。

なぜ第1、第2水準に分けられているかというと、これは使用頻度によるわけだ。たとえば第2水準には國 (国)、學 (学) などの旧字体や、瑟齋、𪚩、𪚪、𪚫、𪚬などの、見たことも聞いたこともないような漢字が並んでいるわけである。それはともかく、一言注意しておかなければならないのは、第1水準が (記号は別として) 「読み」ごとに分類されているの



に対して、第2水準は、部首（偏や冠など）ごとに分類されているということである。第2水準の漢字には、読み方も分からないようなものが多いから、妥当な気もするが、読み方も分からないような漢字をどう使うのであろうか。ま、どーでもいいけどね。

②次に区点コードである。

区点コードは JIS コードを少しアレンジしただけのものである。すなわち、JIS の上位、下位それぞれから  $20_{\text{H}}$  を引いて、10 進数で表し、もう 1 回くっつけ直したものである。だから、たとえば JIS コードの  $4\text{B}7\text{A}_{\text{H}}$ （蘭）は、

$$4\text{B}_{\text{H}} - 20_{\text{H}} = 2\text{B}_{\text{H}} = 43$$

$$7\text{A}_{\text{H}} - 20_{\text{H}} = 5\text{A}_{\text{H}} = 90$$

ということになり、区点コードは「4390」となるのである。はっきり言うが、これは4桁の10進数ではなく、「2桁の10進数が二つ」なのである。

この区点コードというやつは、ただ単に入力する際に A~F を押さなくて済むというぐらゐの意味しかない。これはまだ音訓変換さえもなかったような大昔に、漢字の入力方法として考え出されたものらしい。昔はコンピュータで漢字を使うのは大苦勞だったのである。

③シフト JIS 漢字コード、

このシフト JIS というものは、JIS コードの短所を補おうとしてできたものである（結果的には補えたのかどーだか知らない）。

たとえば JIS コードの  $4141_{\text{H}}$ （疏）を、普通の方法でプリントしようとする、と、 $41_{\text{H}}$  は ASCII コードでは「A」であるから、画面には「AA」と表示されることになる。そのようなことになっては、どーしょーもないので、JIS コードを使う場合にはどうしても「漢字 IN/OUT」という、コントロールコードが必要になってくる。つまり、この場合だと、

（漢字 IN）、 $41_{\text{H}}$ 、 $41_{\text{H}}$ 、（漢字 OUT）

の6バイトのデータで「疏」1文字を表現することになる。

それでは不便だということでできたのがシフト JIS コードである。こいつは、「第1バイトが  $81_{\text{H}} \sim 9\text{F}_{\text{H}}$  もしくは  $\text{E}0_{\text{H}} \sim \text{E}\text{F}_{\text{H}}$  で、なおかつ、第2バイトが  $40_{\text{H}} \sim 7\text{E}_{\text{H}}$  もしくは  $80_{\text{H}} \sim \text{F}\text{C}_{\text{H}}$  の4桁の16進数を漢字とみなす」というものである。ダサイことに第1バイトの範囲に隙間がある。これは、半角のカタカナの ASCII コード  $\text{A}0_{\text{H}} \sim \text{D}\text{F}_{\text{H}}$  だからなのである。シフト JIS を考え出した人は、半角のカタカナと共存させたかったようである。もっとも、カタカナの代わりにグラフィックキャラクタの一部が使えなくなる。しかし、普通の文字（半角のアルファベットなど）と漢字をほとんど同じように扱えるというメリットが出てくるわけである。

ちなみに先程の「疏」は、シフト JIS コードだと、

$9160_{\text{H}}$

となり、「 $91_{\text{H}}$ 、 $60_{\text{H}}$ 」の2バイトのデータだけで「疏」を表現できることになる。第1バイトが  $91_{\text{H}}$  であることが、漢字 IN の代わりになり、第2バイトの次に暗黙の漢字 OUT があるわけだ。つまり、これによって「漢字 IN/OUT」が不要になる。

そこでこの三つにコードの間の変換プログラムが必要になってしまうわけだ。というと



ここで、

リスト 3-1 : JIS ⇄ シフト JIS

リスト 3-2 : JIS ⇄ 区点

の変換プログラムとなっている。

リスト 3-1 JIS⇄シフトJIS

```
100 DEFSNG J
110 FOR J=&H8100 TO &H9FFF:GOSUB150:NEXT:'すべての漢字を走る
120 FOR J=&HE000 TO &HEB00:GOSUB150:NEXT
130 END
140 '
150 LOCATE0,0:PRINTCHR$(J):P$=SCRNS$(0,0,2):'一度表示してから
160 JI$=JIS$(P$):GOSUB 260:'変換し
170 IF EC THEN STOP
180 IF SJ$<>HEX$(ASC(P$)) THEN STOP:'チェックする
190 '
200 SJ$=HEX$(ASC(P$)):GOSUB 420:'変換し
210 IF EC THEN STOP
220 IF JI$<>JIS$(P$) THEN STOP:'チェックする
230 RETURN
240 '
250 'JIS->SHIFT JIS
260 EC=0
270 W$=MKI$(VAL("&H"+JI$))
280 JI1=ASC(RIGHT$(W$,1)):JI2=ASC(LEFT$(W$,1))
290 '
300 IF (JI1<&H21) OR (&H7E<JI1) THEN EC=1:RETURN
310 IF (JI2<&H21) OR (&H7E<JI2) THEN EC=1:RETURN
320 SJ1=INT((JI1-&H21)/2)+&H81
330 IF JI1>=&H5F THEN SJ1=SJ1+(&HE0-&H9F-1)
340 '
350 IF JI1 AND 1 THEN SJ2=&H40:GOTO 360 ELSE SJ2=&H9F:GOTO 370
360 IF (&H60<=JI2) THEN SJ2=SJ2+1
370 SJ2=SJ2+JI2-&H21
380 SJ=SJ1*256+SJ2:SJ$=HEX$(SJ)
390 RETURN
400 '
410 'SHIFT JIS->JIS
420 EC=0
430 W$=MKI$(VAL("&H"+SJ$))
440 SJ1=ASC(RIGHT$(W$,1)):SJ2=ASC(LEFT$(W$,1))
450 '
460 IF (&H81<=SJ1) AND (SJ1<=&H9F) THEN 490
470 IF (&HE0<=SJ1) AND (SJ1<=&HEF) THEN SJ1=SJ1-&HE0+&H9F+1:GOTO 490
480 EC=1:RETURN
490 JI1=INT((SJ1-&H81)*2)+&H21
500 '
510 IF (&H40<=SJ2) AND (SJ2<=&H7E) THEN 540
520 IF (&H80<=SJ2) AND (SJ2<=&HFC) THEN SJ2=SJ2-1:GOTO 540
530 EC=1:RETURN
540 IF SJ2>=&H9E THEN JI1=JI1+1:SJ2=SJ2-&H9E+&H40
550 JI2=SJ2-&H40+&H21
560 '
570 JI$=HEX$(JI1)+HEX$(JI2)
580 RETURN
```

リスト 3-1 は、JI\$に JIS コードを入れ、GOSUB 260 とすると、SJ\$にシフト JIS コードが入って返ってくる。その逆が GOSUB 420 である。ただし、どちらの場合でも、EC=1 でリターンしたなら、あり得ないコードを変換しようとしたということである（つまりエラー）。



リスト 3-2 JIS↔区点

```

100 DEFSNG J
110 FOR J=&H8100 TO &H9FFF:GOSUB150:NEXT
120 FOR J=&HE000 TO &HEB00:GOSUB150:NEXT
130 END
140 '
150 LOCATE0,0:PRINT CHR$(J):P$=SCRN$(0,0,2)
160 JIS$=JIS$(P$):GOSUB 240
170 IF KT$<>KTN$(P$) THEN STOP
180 '
190 KT$=KTN$(P$):GOSUB 320
200 IF JIS$<>JIS$(P$) THEN STOP
210 RETURN
220 '
230 'JIS->KUTEN
240 W$=MKIS(VAL("&H"+JIS$))
250 JI1=ASC(RIGHT$(W$,1)):JI2=ASC(LEFT$(W$,1))
260 KT1=JI1-&H20
270 KT2=JI2-&H20:'単にバイアスをかけるだけ
280 KT$=RIGHT$(STR$(10000+KT1*100+KT2),4)
290 RETURN
300 '
310 'KUTEN->JIS
320 W=VAL(KT$)
330 KT1=W ¥ 100:KT2=W MOD 100
340 JI1=KT1+&H20
350 JI2=KT2+&H20:'単にバイアスをかけるだけ
360 JIS$=HEX$(JI1*256+JI2)
370 RETURN

```


リスト 3-2 は SJ\$ でなく、KT\$ (区点コード) になっただけである。

細かなプログラムの動きは、図 3-1 の「コード相関図」を見て納得していただきたいと思うしだいである。

図 3-1 コード相関図

① シフト JIS コードと JIS 漢字コードの関係

mm \ nn	00	3F	40	7E	80	9E	9F	FC	FF
00									
81			2121 2321	215F 235F	2160 2360	217E 237E	2221 2421	277E 277E	
9F			5D21	5D5F	5D60	5D7E	5E21	5E7E	
E0			5F21	5F5F	5F60	5F7E	6021	607E	
EF			7D21	7D5F	7D60	7D7E	7E21	7E7E	
FC									
FF									

mm, nn はそれぞれシフト JIS コードの第 1, 第 2 バイトを示す  
 は未使用領域



B シフトJISコードと区点コードの関係

	00	3F	40	7E	7F	80	9E	9F	FC	FF
mm	00									
nn	81		0101 0301	0163 0363		0164 0194 0364 0394	0201 0294 0401 0494			
	9F		6101	6163		6164 6194	6201 6294			
	E0		6301	6363		6364 6394	6401 6494			
	EF		9301	9363		9364 9394	9401 9494			
	FC									
	FF									

mm, nnはそれぞれシフトJISコードの第1, 第2 バイトを示す  
 □ は未使用領域

## 表示するのである

X1/turbo で漢字を表示するには、

- ① 漢字 VRAM を使う (turbo)
- ② グラフィック画面に描く

の2とおりの方法があるわけだ。①の場合は、turbo の漢字 VRAM に適当な値をシュポポポンと数バイト書き込むだけ——などと思ったら大間違い。漢字 VRAM に書き込むべきデータ (漢字 ROM アドレスという) の計算が結構面倒なのである。

まず、JIS コードは2バイトであるから、すなわち16ビットなわけだ。しかし、よく見ると、第1, 第2 バイトともに、 $21_{\text{H}} (&B00100001) \sim 7E_{\text{H}} (&B01111110)$  の範囲である。すなわち、どちらのバイトでも、**第7ビットが0**なのだ。よって、JIS コードは実際のところ  $7 \times 2 = 14$  ビットで済んでしまうということになる。それに対して、turbo の漢字 ROM アドレスというのは、I/O マップの漢字 VRAM の所を見てもらえば分かるが、13ビット ( $8 + 5$  ビット) の値なのである。まだ1ビットの差が残っているが、これはいかなるわけかという、JIS コードの  $2821_{\text{H}} \sim 2F7E_{\text{H}}$  と、 $7425_{\text{H}} \sim 7E7E_{\text{H}}$  が空いているからなどの理由によるのである (ここらへんのところは、JIS の改訂などがあるのでちよいと違う部分もある)。

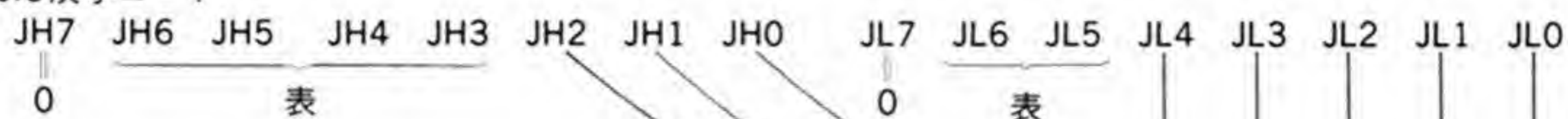
というわけで、**図 3-2** が JIS コード→漢字 ROM アドレス変換の方法を図示したもの、**リスト 3-3** が変換を実行するプログラムである。少し付け加えておくが、要するに、漢字 ROM アドレスの下位8ビット ( $3000_{\text{H}} \sim 37FF_{\text{H}}$  の I/O 空間に OUT するやつ) は、単  
 68 試験に出る X1



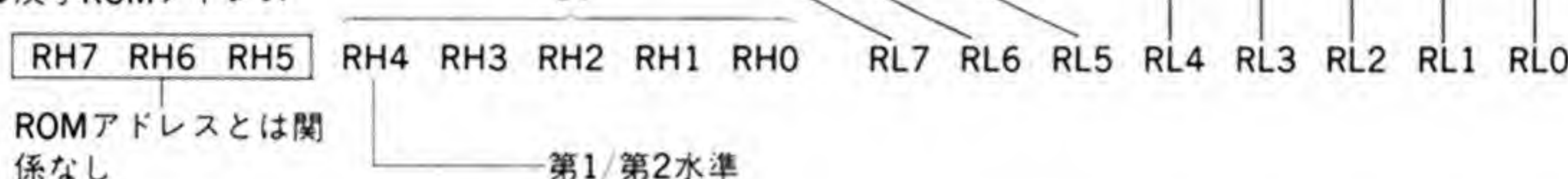
純に決定するのだが、3800<sub>H</sub>～3FFF<sub>H</sub>の漢字 VRAM の方のデータは図 3-2 の下の表を見なければならぬということである。注意深く見れば規則性が見えてくるであろう。

図 3-2 turbo のJIS コード⇔漢字 ROM アドレスの関係

●JIS漢字コード



●漢字ROMアドレス



		JIS漢字コード					漢字ROMアドレス					JIS漢字コード
		JH6	JH5	JH4	JL6	JL5	RH4	RH3	RH2	RH1	RH0	
第1水準		0	1	0	0	1	↔	0	0	0	0	JH3
		0	1	0	1	0	↔	0	0	0	1	JH3
		0	1	0	1	1	↔	0	0	0	0	JH3
		0	1	1	0	1	↔	0	0	1	0	JH3
		0	1	1	1	0	↔	0	0	1	1	JH3
		0	1	1	1	1	↔	0	1	0	0	JH3
		1	0	0	0	1	↔	0	1	0	1	JH3
		1	0	0	1	0	↔	0	1	1	0	JH3
		1	0	0	1	1	↔	0	1	1	1	JH3
第2水準		1	0	1	0	1	↔	1	0	0	0	JH3
		1	0	1	1	0	↔	1	0	0	1	JH3
		1	0	1	1	1	↔	1	0	1	0	JH3
		1	1	0	0	1	↔	1	0	1	1	JH3
		1	1	0	1	0	↔	1	1	0	0	JH3
		1	1	0	1	1	↔	1	1	0	1	JH3
		1	1	1	0	1	↔	1	1	1	0	JH3
		1	1	1	1	0	↔	1	1	1	1	JH3
		1	1	1	1	1	↔	1	1	1	0	JH3

リスト 3-3 turbo 用 JIS⇔漢字 ROM アドレス変換

```

100 DEFSNG J
110 FOR J=&H8100 TO &H9FFF:GOSUB 150:NEXT
120 FOR J=&HE000 TO &HEAFF:GOSUB 150:NEXT
130 END
140 '
150 LOCATE 0,0:PRINT CHR$(J):J$=SCRN$(0,0,2)
160 IF J$="※" THEN 290
170 JI$=JIS$(J$):GOSUB 320      : 'JIS -> ROM
180 IF EC THEN STOP
190 IF RL<>INP(&H3000) THEN STOP
200 IF RH<>(INP(&H3800) AND &H1F) THEN STOP
210 OUT &H3804,RH OR &H80
220 OUT &H3805,RH OR &HC0
230 OUT &H3004,RL:OUT &H3005,RL

```



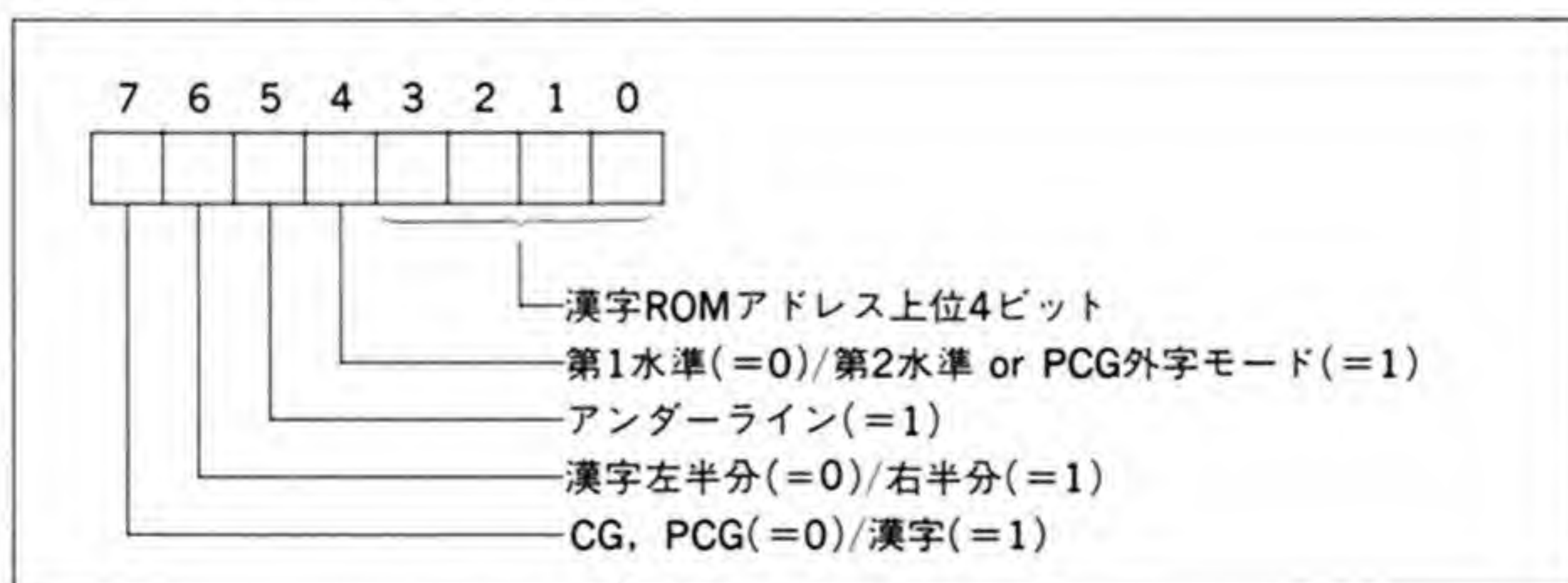
```

240 '
250 RL=INP(&H3000)
260 RH=INP(&H3800) AND &H1F
270 JI$="":GOSUB 490          : 'ROM -> JIS
280 IF JI$<>JIS$(J$) THEN STOP
290 RETURN
300 '
310 'JIS->ROM
320 EC=0
330 JI=VAL("&H"+JI$)
340 RL=(JI AND &B11111) OR ((JI ¥ 8) AND &B11100000)
350 RW1=(JI ¥ &H1000) AND 7      : 'JH6,JH5,JH4
360 RW2=(JI ¥ &H800) AND 1      : 'JH3
370 RW3=(JI ¥ &H20) AND 3       : 'JL6,JL5
380 ON RW1 GOTO 390,400,450,450,450,450,410
390 EC=1:RETURN
400 RH=&B0 OR RW2      :GOTO 420
410 RH=&B11100 OR RW2
420 IF RW3=2 THEN RH=RH OR 2
430 IF RW3=3 THEN RH=RH XOR 1
440 RETURN
450 RH=((RW1-3)*3+1+RW3)*2 OR RW2
460 RETURN
470 '
480 'ROM->JIS
490 EC=0
500 JI2=RL AND &B11111      : 'RL4 - RL0
510 JI1=(RL ¥ &H20) AND &B111 : 'RL7 - RL5
520 IF RH<4 THEN 580        : '2121H - 277EH
530 IF RH>&H1C THEN 590     : '7021H - 777EH
540 JI1=JI1 OR (((RH ¥ 2)+7)¥3)*&H10)
550 JI1=JI1 OR (RH AND 1)*8
560 JI2=JI2 OR (((((RH ¥ 2)+1)MOD3)+1)*&H20)
570 GOTO 650
580 JI1=JI1 OR &B100000 :GOTO 600
590 JI1=JI1 OR &B1110000:GOTO 600
600 RW=RH AND 3
610 IF RW=0 THEN JI2=JI2 OR &B100000:GOTO 650
620 IF RW=2 THEN JI2=JI2 OR &B100000:GOTO 650
630 IF RW=1 THEN JI2=JI2 OR &B1100000:GOTO 650
640 EC=1:RETURN
650 JI$=HEX$(JI1)+HEX$(JI2):RETURN

```

210～230 行のルーチンは変換を実行した後で、実際に漢字 VRAM へそのデータを OUT するものである。図 3-3 にあるように、左側/右側の指定なども行なっていることに注意である。

図 3-3 漢字 VRAM の各ビットの機能





## さらに表示するのである

②のグラフィックに表示する場合をやるのである。これは X1 と turbo の場合に分かれるのであるが、turbo の場合は、第 2 章でやった「CG (漢字 ROM) の読み出し」を実行した後で、そのデータをグラフィックに書き込んでしまえばよいわけであるから、さっさと X1 の場合に行ってしまうわけである。

X1 における漢字 ROM といえば、CZ-8KR, CZ-8BK2 である (X1F/G の model 20 には標準で内蔵している)。で、そいつのアクセス方法が主題になるわけだったりする。

漢字 ROM の読み出しはリスト 3-4 である。これは JIS コードを入力して、対応する漢字を画面の左上に表示するプログラムである。説明すると、

120~140 行: JIS コードを入力し、1 バイト目 (JI1) と 2 バイト目 (JI2) に分離する。

150~170 行: ROM に JI1 と 0 を OUT し、2 バイト読む。

リスト 3-4 X1 用漢字 ROM 読み出しプログラム

```
100 INIT:PALET 1,7:CLS4
110 '
120 LOCATE 0,5:INPUT JIS      :'INPUT JIS CODE
130 JI1=VAL("&H"+LEFT$(JIS,2)) :'JIS HIGH
140 JI2=VAL("&H"+RIGHT$(JIS,2)) :'JIS LOW
150 OUT &HE80,JI1
160 OUT &HE81,0
170 GOSUB 330      :'READ 2 BYTE FROM ROM
180              :'GET R1 (FROM TABLE)
190 '
200 R=R1*&H100+(JI2-&H20)*&H10
210 OUT &HE82,0
220 OUT &HE80,ASC(LEFT$(MKIS(R),1)):'SET LOW ADDR.
230 OUT &HE81,INT(R/256)           :'SET HIGH ADDR.
240 FOR I=0 TO 15
250   GOSUB 330 :'READ 2 BYTE FROM ROM
260   LP=R1:RP=R2
270   IF I<8 THEN AD=&H4000+I*&H800 ELSE AD=&H4050+(I-8)*&H800
280   OUT AD,LP:OUT AD+1,RP      :'DRAW PATTERN
290 NEXT
300 GOTO 120      :'LOOP
310 '
320 'READ FROM ROM
330 OUT &HE82,1
340 R1=INP(&HE80)
350 R2=INP(&HE81)
360 OUT &HE82,0
370 RETURN
```

これにより、「漢字 ROM 内のテーブル」を使って、本当の「漢字パターンの ROM 内アドレス」が分かるのである。これについては、表 3-1 を見ていただきたい。これは参考文献 1 からの引用である。

結局そのようにして得られた ROM 内アドレス (200 行の変数 R) を、もう一度 E80<sub>H</sub>, E81<sub>H</sub> 番地に指定し、漢字パターンの読み出しにかかるのが、210~290 行である。一度に読み出すのは 2 バイトで、それが 16×16 の漢字の横 1 ラインになる。結局それを 16 回繰り返すことによって、めでたく一つの漢字のパターンとなるのである。なお、270 行、280 行



表 3-1 漢字 ROM (CZ-8KR, 8KR2)の I/O ポート

I/Oアドレス	入出力	機 能
0E82 <sub>H</sub>	出 力	00 <sub>H</sub> 出力……ROMからのデータ読み出し終了 01 <sub>H</sub> 出力……ROMからのデータ読み出し開始
0E81 <sub>H</sub> 0E80 <sub>H</sub>	入出力	<div> <div> I/O(0E81<sub>H</sub>)←00<sub>H</sub>  I/O(0E80<sub>H</sub>)←JIS漢字コード上位バイト </div> <div>のとき</div> </div> データ読み出し時には、 I/O(0E81 <sub>H</sub> )からは00 <sub>H</sub> I/O(0E80 <sub>H</sub> )からはROM内アドレス上位バイト が得られる。
		<div> <div> I/O(0E81<sub>H</sub>)←ROM内アドレス上位バイト(≠00<sub>H</sub>)  I/O(0E80<sub>H</sub>)←ROM内アドレス下位バイト </div> <div>のとき</div> </div> データ読み出し時には、 I/O(0E81 <sub>H</sub> )からは、右部分フォントパターン16バイト分 I/O(0E80 <sub>H</sub> )からは、左部分フォントパターン16バイト分 が得られる。

は簡便にグラフィックの漢字を表示するためのルーチンである。

ところで、先に言うておくが、機械語で漢字 ROM にアクセスする場合は、E82<sub>H</sub>へ01<sub>H</sub>をOUTしてからデータを読み始めるまでの間（リスト 3-4 の 335 行にあたる）に 3 μs 以上ウェイトが必要ということになっている（リスト 3-5 参照）。

というところで、駄目押しとして、漢字 ROM にアクセスする機械語プログラムがリスト 3-5 で、それを BASIC から使っているのがリスト 3-6 である。当然なことであるが turbo では動かない。

リスト 3-5 機械語で漢字 ROM にアクセス

		.280
		.PHASE 0FE00H
		;
		;KANJI PATTERN READ
		;DE -> JIS-HIGH,JIS-LOW
FE00	EB	START1: EX DE,HL
		;
FE01	56	LD D,(HL)
FE02	23	INC HL
FE03	5E	LD E,(HL)
FE04	18 0F	JR KREAD
		;
		;=====
		;KANJI PATTERN DRAW
		;DE -> JIS-HIGH,JIS-LOW,X,Y
		;!!! X,Y = 16 bits !!!
FE06	EB	START2: EX DE,HL
		;
FE07	56	LD D,(HL)
FE08	23	INC HL
FE09	5E	LD E,(HL)
FE0A	23	INC HL
		;
FE0B	4E	LD C,(HL)
FE0C	23	INC HL
FE0D	46	LD B,(HL)
FE0E	23	INC HL ;GET X
		;
FE0F	7E	LD A,(HL)
FE10	23	INC HL
FE11	66	LD H,(HL) ;GET Y
FE12	6F	LD L,A



FE13	18 16	;	JR	KDRAW	
		;			
		*****			
		;DE=JIS CODE:D=HIGH,E=LOW			
		;READ KANJI PATTERN			
		;			
FE15	CD FE6F	KREAD:	CALL	SETROM	
		;			
FE18	21 FED0		LD	HL,PATA ;PATTERN AREA	
FE1B	3E 10		LD	A,16	
FE1D	08	KREADL:	EX	AF,AF' ;SAVE COUNTER	
FE1E	CD FE8F		CALL	RROM	
FE21	72		LD	(HL),D ;GET LEFT	
FE22	23		INC	HL	
FE23	73		LD	(HL),E ;GET RIGHT	
FE24	23		INC	HL	
FE25	08		EX	AF,AF' ;CHECK COUNT	
FE26	3D		DEC	A	
FE27	C2 FE1D		JP	NZ,KREADL	
FE2A	C9		RET		
		;			
		;DE=JIS CODE:D=HIGH,E=LOW			
		;BC=X,HL=Y			
		;DRAW KANJI PATTERN			
		;			
FE2B	D5	KDRAW:	PUSH	DE ;SAVE JIS CODE	
FE2C	CD FEA3		CALL	XYADDR ;GET VRAM	
FE2F	44		LD	B,H	
FE30	4D		LD	C,L ;COPY ADDR.	
		;BC=G.ADDR.			
FE31	D9		EXX		
FE32	D1		POP	DE ;GET JIS	
FE33	CD FE6F		CALL	SETROM	
		;			
FE36	01 0E82		LD	BC,0E82H	
FE39	D9		EXX		
		;			
FE3A	AF		XOR	A	
FE3B	08		EX	AF,AF'	
FE3C	3E 10		LD	A,16 ;COUNTER	
		;			
FE3E	08	KDRAWL:	EX	AF,AF'	
FE3F	D9		EXX		
		;BC=0E82H,A=0			
FE40	3C		INC	A	
FE41	ED 79		OUT	(C),A ;BEGIN READ	
FE43	0D		DEC	C	
FE44	0D		DEC	C	
FE45	00		NOP	;3 us	
		;			
FE46	ED 78		IN	A,(C) ;LEFT	
FE48	0C		INC	C	
FE49	D9		EXX		
FE4A	ED 79		OUT	(C),A ;DRAW LEFT	
FE4C	03		INC	BC	
FE4D	D9		EXX		
		;			
FE4E	ED 78		IN	A,(C) ;RIGHT	
FE50	D9		EXX		
FE51	ED 79		OUT	(C),A ;DRAW RIGHT	
FE53	D9		EXX		
		;			
FE54	AF		XOR	A	
FE55	0C		INC	C	
FE56	ED 79		OUT	(C),A ;END READ	
		;BC=0E82H,A=0			
FE58	D9		EXX		
		;BC=G.ADDR.			
		;DOWN 1 LINE			
FE59	0B		DEC	BC ;BACK ADDR	
FE5A	21 0800		LD	HL,800H	
FE5D	B7		OR	A	
FE5E	ED 4A		ADC	HL,BC ;CHECK SIGN FLAG	
FE60	F2 FE67		JP	P,OKDN ;OK(ONLY BLUE)	
		;			
FE63	21 C850		LD	HL,4050H-7800H	
		;IF WIDTH 40			
		;	LD	HL,4028H-7800H	
FE66	09		ADD	HL,BC	
FE67	44	OKDN:	LD	B,H	



FE68	4D	LD	C,L	
FE69	08	EX	AF,AF'	
FE6A	3D	DEC	A	
FE6B	C2 FE3E	JP	NZ,KDRAWL	
FE6E	C9	RET		
;=====				
FE6F	7B	SETROM: LD	A,E	;LOW
FE70	D6 20	SUB	20H	
FE72	6F	LD	L,A	;L=LOW-20H
FE73	01 0E80	LD	BC,0E80H	
FE76	ED 51	OUT	(C),D	;E80 <- JISH
FE78	0C	INC	C	
FE79	AF	XOR	A	
FE7A	ED 79	OUT	(C),A	;E81 <- 00H
FE7C	CD FE8F	CALL	RROM	;GET FROM TABLE
FE7F	26 00	LD	H,00H	
FE81	29	ADD	HL,HL	
FE82	29	ADD	HL,HL	
FE83	29	ADD	HL,HL	
FE84	29	ADD	HL,HL	;HL=HL*16
FE85	19	ADD	HL,DE	
FE86	01 0E80	LD	BC,0E80H	
FE89	ED 69	OUT	(C),L	;E80 <- LOW
FE8B	0C	INC	C	
FE8C	ED 61	OUT	(C),H	;E81 <- HIGH
FE8E	C9	RET		
FE8F	01 0E82	LD	BC,0E82H	
FE92	3E 01	LD	A,1	
FE94	ED 79	OUT	(C),A	;BEGIN READ
FE96	0D	DEC	C	
FE97	0D	DEC	C	
FE98	00	NOP		;3 us
FE99	ED 50	IN	D,(C)	;LEFT
FE9B	0C	INC	C	
FE9C	ED 58	IN	E,(C)	;RIGHT
FE9E	AF	XOR	A	
FE9F	0C	INC	C	
FEA0	ED 79	OUT	(C),A	;END READ
FEA2	C9	RET		
;ADDR=4000H+(X1>>3)+((Y1 & 7)<<11)+(Y1>>3)*80				
;ADDR=4000H+(BC/8)+((L AND 7)<<11)+(HL/8)*80				
;HL=Y1,BC=X1,BREAKS HL,A,BC,DE				
;return HL=addr,A=mask,D=count				
FEA3	7D	XYADDR: LD	A,L	;SAVE L
FEA4	CD FEC3	CALL	DIV8	;HL=HL/8
FEA7	54	LD	D,H	
FEA8	5D	LD	E,L	;DE=HL
FEA9	29	ADD	HL,HL	
FEAA	29	ADD	HL,HL	;80=16*5
FEAB	19	ADD	HL,DE	;HL=5
FEAC	29	ADD	HL,HL	;10
FEAD	29	ADD	HL,HL	;20
FEAE	29	ADD	HL,HL	;40
FEAF	29	ADD	HL,HL	;80 ;HL=(HL/8)*80
FEB0	E6 07	AND	07H	;A=(L AND 7)
FEB2	87	ADD	A,A	
FEB3	87	ADD	A,A	
FEB4	87	ADD	A,A	;A=((L AND 7)<<3)



FEB5	C6 40	ADD	A,040H	;ADD 4000H
FEB7	57	LD	D,A	
FEB8	1E 00	LD	E,00H	
			;DE=4000H+((L AND 7)<<(3+8))	
FEBA	19	ADD	HL,DE	;LAST 2 & 1ST WERE DONE
FEBB	EB	EX	DE,HL	;HDE=HL (SAVE)
			;	
FEBC	60	LD	H,B	
FEBD	69	LD	L,C	;HL=BC
			;	
FEBE	CD FEC3	CALL	DIV8	;HL=HL/8
FEC1	19	ADD	HL,DE	;BADDR DONE
FEC2	C9	RET		
			;HL has result (address),	
			;	
FEC3	CB 3C	DIV8:	SRL	H
FEC5	CB 1D		RR	L
FEC7	CB 3C		SRL	H
FEC9	CB 1D		RR	L
FECB	CB 3C		SRL	H
FECD	CB 1D		RR	L
FECF	C9		RET	
			;	
FED0		PATA:	DS	32 ;PAT AREA
			;	
			END	

リスト 3-6 BASIC から「リスト 3-5」を使う

100	CLEAR &HFE00
110	MEM\$(&HFE00,16)=HEXCHR\$("EB 56 23 5E 18 0F EB 56 23 5E 23 4E 23 46 23 7E")
120	MEM\$(&HFE10,16)=HEXCHR\$("23 66 6F 18 16 CD 6F FE 21 D0 FE 3E 10 08 CD 8F")
130	MEM\$(&HFE20,16)=HEXCHR\$("FE 72 23 73 23 08 3D C2 1D FE C9 D5 CD A3 FE 44")
140	MEM\$(&HFE30,16)=HEXCHR\$("4D D9 D1 CD 6F FE 01 82 0E D9 AF 08 3E 10 08 D9")
150	MEM\$(&HFE40,16)=HEXCHR\$("3C ED 79 0D 0D 00 ED 78 0C D9 ED 79 03 D9 ED 78")
160	MEM\$(&HFE50,16)=HEXCHR\$("D9 ED 79 D9 AF 0C ED 79 D9 0B 21 00 08 B7 ED 4A")
170	MEM\$(&HFE60,16)=HEXCHR\$("F2 67 FE 21 50 C8 09 44 4D 08 3D C2 3E FE C9 7B")
180	MEM\$(&HFE70,16)=HEXCHR\$("D6 20 6F 01 80 0E ED 51 0C AF ED 79 CD 8F FE 26")
190	MEM\$(&HFE80,16)=HEXCHR\$("00 29 29 29 29 19 01 80 0E ED 69 0C ED 61 C9 01")
200	MEM\$(&HFE90,16)=HEXCHR\$("82 0E 3E 01 ED 79 0D 0D 00 ED 50 0C ED 58 AF 0C")
210	MEM\$(&HFEA0,16)=HEXCHR\$("ED 79 C9 7D CD C3 FE 54 5D 29 29 19 29 29 29 29")
220	MEM\$(&HFEB0,16)=HEXCHR\$("E6 07 87 87 87 C6 40 57 1E 00 19 EB 60 69 CD C3")
230	MEM\$(&HFEC0,16)=HEXCHR\$("FE 19 C9 CB 3C CB 1D CB 3C CB 1D CB 3C CB 1D C9")
240	'
250	WIDTH80:INIT:CLS 4
260	PALET 1,7
270	DEFUSR0=&HFE00
280	DEFUSR1=&HFE06
290	'D\$=USR0(HEXCHR\$("3E22"))
300	'D\$=USR1(HEXCHR\$("3E22")+MKI\$(10)+MKI\$(10))
310	'
320	X=0:Y=0
330	LOCATE 0,0:INPUT "JIS";JIS\$
340	IF LEN(JIS\$)<>4 THEN 330
350	D\$=USR1(HEXCHR\$(JIS\$)+MKI\$(X)+MKI\$(Y))
360	X=X+16:IF X>640-16 THEN X=0:Y=Y+16:IF Y>200-16 THEN STOP
370	GOTO 330

リスト 3-5 の中には二つのプログラムが入っている。FE00<sub>H</sub>番地からの START1 は漢字のパターンを、FEDA<sub>H</sub>番地からの 32 バイトの領域に読み出すサブルーチンである。それに対して、FE06<sub>H</sub>番地からの START2 は指定された位置 (グラフィック) に、漢字を描くルーチンである。このルーチンは手抜きをしてあり、描画は青画面にしかないようになっている。また、普通に考えるなら、「一旦読み出してから描く」のだろうが、それだと面白くないので、「ROM から読み出しながら描く」方法を取っている。

FE00<sub>H</sub>～からのルーチンの使い方は、DE レジスタが指しているアドレスから、JIS コードの上位、下位に入っていることが条件である。FE06<sub>H</sub>～は、さらにその後に X, Y の座標値が入っている必要がある。で、このようにすると BASIC から USR 関数でお手軽に使え



るというわけである。おっと念のために言っておくが、座標として指定する X, Y は Z80 の 16 ビット (2 バイト) 値で、下位、上位の順になっているわけだ。そして、X の方は、8 ドットごとになっている。つまり、(9, 0) でも (10, 0) でも、(8, 0) に表示されてしまうのである。いちいちビットシフトなんかやってられないのである。

さて、サブルーチンをざっと説明しておく。

#### SETROM :

DE レジスタに入っている JIS コードの漢字パターンを読み出すための設定をする。この部分はリスト 3-4 の 130~230 行に相当する部分である。

#### RROM :

漢字 ROM から 2 バイトだけ読み出すサブルーチン。D レジスタが左側のパターン、E レジスタが右側のパターンの相当データを持ってリターンする。FE98<sub>H</sub> 番地の NOP は、その前にある二つの「DEC C」とともに、3  $\mu$ s のウェイトになるわけである。

#### XYADDR :

X, Y 座標からグラフィックアドレスを求めるサブルーチンである。X 座標は 8 ドットおきに限定しているので、なかなか短くなっている。

というわけであるが、最後に付け加えておくと、FE59<sub>H</sub>~FE68<sub>H</sub>のあたりはグラフィックアドレスを 1 ライン下げるルーチンである。座標を青画面に限定しているので、このように短くなっている。

そいでもってリスト 3-6 を実行すると、JIS コードを聞いてくる。で、適当なコードを入れてやるとポコッと表示するわけである。一番気になったのは速度だったのであるが、どうやら非常に速いようである。ただしそれは機械語に入ってからのもので、実際にリスト 3-6 を走らせても BASIC の部分がどうしても遅いのであった。困ったことよ。

以上、漢字について一とおりやったわけである。しかしながら、この章でやったことは最小限のことだけである。理想を言えば、せめて単漢字変換ぐらいやっておきたいわけであるが、ソフト的なことはまったく棚に上げてしまったわけである。

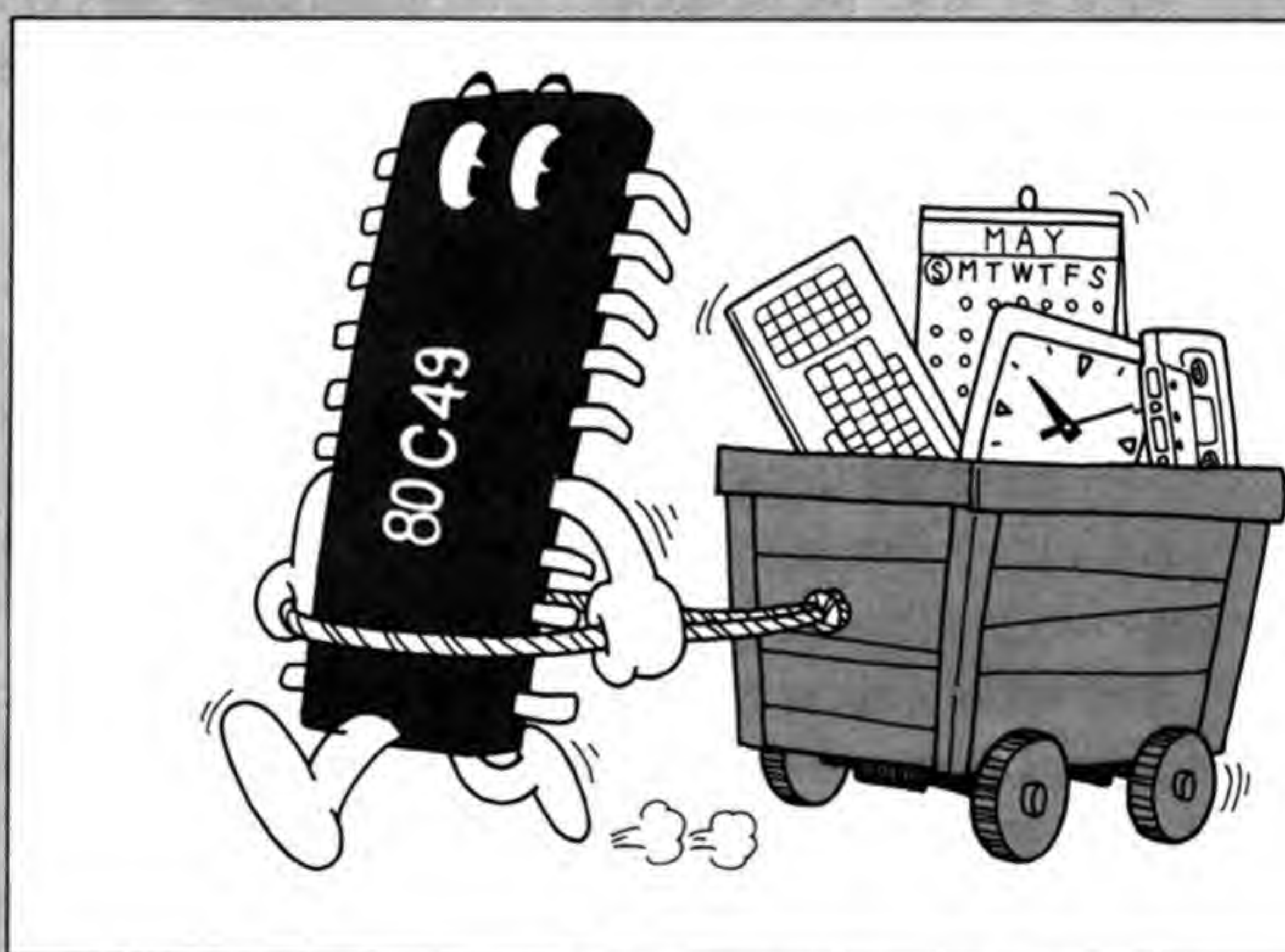


第

4

章

# サブCPU



サブCPUのおかげなのである



## ■第4章

# サブCPUのおかげなのである・・・

その存在が意識されることは少ないが、X1では、カセットのコントロール、キーボードからの入力、時計、カレンダーはサブCPUの縄張りになっているのである。てなわけで、この章ではその裏方さんのサブCPUをやるのである。

## 80C49なのである

まずはサブCPUそのものについて紹介しておこう。X1に使われているサブCPUは、80C48、80C49という2種類の8ビットワンチップマイコンである。これらの石はインテル社の発表した8048、8049のC-MOS版なので、型番の間に「C」が入っている。ちなみにC-MOSの特徴の一つは消費電力が極めて少ないということである。そのことによってX1の電源がOFFになっていてもサブCPUは生きていて、キーボードからテレビをコントロールできるのである。さもなくばX1は電気食い虫になり下がってしまうのである。

それはさておき、80C48、80C49について基礎教養講座を開くのである。

ワンチップマイコンというものは、主に制御用に使われているLSIである。最近では家庭電気器具にマイコンを組み込むのがはやっていて、CMなどでもしょっちゅう自慢しているからおなじみであろう。

ワンチップマイコンには4ビットのものと8ビットのものがある（そのうち技術をもて余したチップメーカーが16ビットや32ビットのものを作るかもしれない）。このビット数の違いは、パソコン用のCPUの8ビット、16ビット、32ビットと同じようなものである。早い話が処理能力が違うのだ。80C48と80C49はめでたいことに8ビットである。うっかり4ビットCPUにキーボードをまかせたりしたら、ゲームをするときのキー操作がFM-7のようになっていたかもしれない。鶴亀鶴亀。

さて、80C48と80C49の違いであるが、これら二つのマイコンは基本的に同じなのである。序列は80C49の方が上位（高機能）である。これらのマイコンは、一つのチップの中に、CPU部分と、ROM、RAMを持っていて、基本的にプログラムや定数データはすべてチップ内のROMに入る。いわばクリーンコンピュータの完全な逆なのである。チップ内のRAMは主に変数やワークエリアとして使われる。二つのマイコンの違いはこれらのメモリの容量で、80C48のROM：1Kバイト、RAM：64バイトに対して、80C49ではそれぞれが倍になっている。ただしワンチップマイコンといっても外部にROMやRAMを付けることも可能で、そのようにすれば両方とも同じ性能にまで拡張できるそうである。詳しくは参考文献9を見ていただきたい。



## サブCPUの実態なのである

X1でのサブCPUの使われ方は、80C49が本体の中にあり、キーボード、カセットデッキ、カレンダー、時計( $\mu$ PD1990というLSI)とZ80との間をとりもっている。X1のキーボードの中には80C48が入っており、押されたキーのデータを80C49がシリアル転送している。これは80C48(キーボード)から80C49(本体)への一方通行である。すなわち間に赤外線を送受信装置などを挟めば、かなり簡単にワイヤレスになるはずである。さて、turboのキーボードではちよいと違う。80C48ではなく、80C49が使われているのである。これは「Bモード」なんちゅうものがキーボードに付いたりなどしたため、80C48では処理能力が足りなくなったためだと思われる。

turboでBモードという機能が付いたことから想像できるように、実は本体内のサブCPUの80C49も、X1とturboでは違うのである。正確に言えば、内部のROM(マスクROM)に書かれているプログラムが違うのである。さらに、turboのmodel 20/30においてはそれだけではなく、データレコーダ(CZ-8RL1)をつなぐための変更もある。8RL1にも80C49が内蔵されており、インテリジェントになっている(確認はしていないが、専用ディスプレイにも入っていると思う)。turboのmodel 20, 30のサブCPUは、8RL1の80C49同士とシリアル通信を行なっているのである。これは8RL1をつなぐことができるX1F model 20でも同じことである。すなわち、本体内のサブCPUは、

- 1) X1 turbo model 20, 30, 40 (II, III, Zも含む)
- 2) X1 turbo model 10
- 3) X1F, G model 20
- 4) その他

の4種で、それぞれ別の仕様になっているのである。もっと細かい部分や型番なども違う可能性があるが、少なくとも上記の4種は機能が違っていることは確かである。8RL1をつなげるようにした部分の変更はZ80側からはまったく見えない。すなわち、サブCPUがその部分の「違い」を完全に吸収しているのである。てなわけで、周辺機器の構成の変化にも対応できる点が、サブCPUを使うメリットでもあるのだ。

## さて、使い方である

サブCPUを使うには基本的に2とおりの方法がある。それはサブCPUからデータを受け取るか、それともサブCPUにデータを渡すかの違いである。ただし、どちらの場合でも最初はZ80から80C49へコマンドを送ってやらなければならない。80C49はそのコマンドを見て、自分がデータをZ80に送るのか、それともZ80からデータを受け取るのかを判断するのである。なお、それぞれの場合の受け渡しをするデータ数(バイト数)も決まっているので、その数を守らないと正常な動作をしなくなってしまうので注意が必要である。というところで表4-1にサブCPUのコマンド表を示す。



表 4-1 サブ CPU コマンド一覧

コマンド	動 作	データ転送方向	データ数 (バイト数)	備 考
E3 <sub>H</sub>	ゲームキーデータ読み出し	Z80←80C49	3	turbo のみ。キーボードのモードが A ならデータは 00 <sub>H</sub>
E4 <sub>H</sub>	キー入力割り込みベクタセット	Z80→80C49	1	ベクタを 00 <sub>H</sub> にすると、割り込みは起きない
E6 <sub>H</sub>	キーデータ読み出し	Z80←80C49	2	データはファンクション部、ASCII コードの順
E7 <sub>H</sub>	TVコントロール	Z80→80C49	1	特になし
E8 <sub>H</sub>	TVコントロール読み出し	Z80←80C49	1	TVに最後に送られたコードを読み出す
E9 <sub>H</sub>	カセットデッキコントロール	Z80→80C49	1	データは、CMT=〜に対応
EA <sub>H</sub>	カセットデッキ状態読み出し	Z80←80C49	1	データは、〜=CMTに対応
EB <sub>H</sub>	カセットセンサー読み出し	Z80←80C49	1	データは、〜=CMT(Δ)に対応
EC <sub>H</sub>	カレンダーセット	Z80→80C49	3	特になし
ED <sub>H</sub>	カレンダー読み出し	Z80←80C49	3	特になし
EE <sub>H</sub>	時計セット	Z80→80C49	3	特になし
EF <sub>H</sub>	時計読み出し	Z80←80C49	3	特になし
D0 <sub>H</sub> ~D7 <sub>H</sub>	タイマ(0~7)セット	Z80→80C49	6	特になし
D8 <sub>H</sub> ~DF <sub>H</sub>	タイマ(0~7)読み出し	Z80←80C49	6	特になし

では、例によってサンプルプログラムを使いながら順に解説するのであった。

#### ① サブ CPU と会話するにも礼儀あり

第 0 章の I/O マップを見ても分かるように、サブ CPU との会話は I/O 空間の 1900<sub>H</sub> 番地を使って行なうのである。しかし、ただ単に 1900<sub>H</sub> 番地にデータを IN/OUT すればよいというものではない。サブ CPU にもサブ CPU の都合というものがあり、おのずと礼儀を守らなければならないのである。早い話が 1A01<sub>H</sub> 番地のビット 6 が 1 (セットされている) ならば、サブ CPU は「今は忙しいからデータは受け取れません」といっているのである。また同じく 1A01<sub>H</sub> 番地のビット 5 が 1 (セットされている) ならば、「今はあんたに渡すデータはないです」と言っているのである。すなわち、

##### 1A01<sub>H</sub> 番地のビット 6

0 ならば Z80 → 80C49 へ 1 バイト

1 ならばダメ

##### 1A01<sub>H</sub> 番地のビット 5

0 ならば 80C49 → Z80 へ 1 バイト

1 ならばダメ

なのである。

そこでリスト 4-1 を見ていただきたい私なのである。このプログラムを打ち込み間違った場合は暴走したり、キー入力を受け付けられなくなったりするので、実行前に一度 SAVE しておくことをお勧めする。さて、このプログラムはオール BASIC である。やっていることはカレンダー (日付) の読み書きなのである。問題の焦点の 80C49 とのデータのやりとりは、

400~430 行が Z80 → 80C49

80 試験に出る X1



リスト 4-1 オール BASIC 版サブ CPU アクセスルーチン

```

100 DATE$="99/02/23"
110 D$=CHR$(&HED)+"...":'GET DATE$
120 GOSUB"GET-DATA"
130 GOSUB"DISPHEX":PRINT
140 '
150 D$=CHR$(&HEC,&H85,&H93,&H18):'DATE$="85/09/18" and WED
160 GOSUB"SET-DATA"
170 END
180 '
190 LABEL"DISPHEX"
200 FORI=2TOLEN(D$)
210   D=ASC(MID$(D$,I,1)):PRINT HEX$(D),
220 NEXT:RETURN
230 '
240 LABEL"GET-DATA"
250 D=ASC(LEFT$(D$,1))
260 GOSUB"Z80->80C49"           :'SET COMMAND
270 FORI=2TOLEN(D$)
280   GOSUB"80C49->Z80"
290   MID$(D$,I,1)=CHR$(D)
300 NEXT:RETURN
310 '
320 LABEL"SET-DATA"
330 D=ASC(LEFT$(D$,1))
340 GOSUB"Z80->80C49"           :'SET COMMAND
350 FORI=2TOLEN(D$)
360   D=ASC(MID$(D$,I,1))
370   GOSUB"Z80->80C49"
380 NEXT:RETURN
390 '
400 LABEL"Z80->80C49"
410 WHILE INP(&H1A01) AND &H40:WEND:'WAIT
420 OUT&H1900,D                  :'SET 1 BYTE
430 RETURN
440 '
450 LABEL"80C49->Z80"
460 WHILE INP(&H1A01) AND &H20:WEND:'WAIT
470 D=INP(&H1900)                :'GET 1 BYTE
480 RETURN

```

450～480 行が 80C49 → Z80

である。ラベルそのものだから別に説明する必要もないのであった。

プログラムではサブ CPU とのやりとりで代表的な二つの形式を示している。まず、100～130 行がデータ入力である。これは、まずサブ CPU にコマンド ED<sub>H</sub>を渡して、「カレンダーのデータをちょうだい」と言うのである。サブ CPU は ED<sub>H</sub>というコマンドを受け取ると、「よっしゃよっしゃ」と言って 3 バイトのデータを送ってくれる。110 行の“...”はその 3 バイトのデータを受け取るためのダミーである。

次に 150～160 行がカレンダーのセットである。最初の EC<sub>H</sub>が、「これからカレンダーをセットするぞ」という意味のコマンドである。サブ CPU はこのコマンドを受け取ると、またもや「よっしゃよっしゃ」と言って 3 バイトのデータを要求してくるのである。そこで、150 行の &H85, &H93, &H18 の 3 個を渡してやるのである。

このサンプルから分かるようにサブ CPU との交信には、

#### ① タイプ 1

1 バイトのコマンドを 80C49 へ送る。

その後 80C49 が数バイトのデータを送ってくる。



## ② タイプ2

1 バイトのコマンドを 80C49 へ送る。

その後 80C49 がさらに数バイトのデータを要求する。

の2とおりがあがあるのだ。①を 240~300 行のサブルーチン、②を 320~380 行のサブルーチンで行なっている。

補足するが、100 行では読み出しがいがあるようにデタラメな日付をカレンダーにセットしている。190~220 行のサブルーチンは D\$ の中身の表示である。

というところで、いきなり凶器攻撃である。実はこのプログラムはぜんぜん正しくないのだ。170 行を「GOTO 110」にしてループさせ、**BREAK** キーだけをハイパーオリピック的にパシパシと叩くと、やがて BASIC が止まってしまうのである。実はこれはキー割り込みのせいなのである。そこで機械語を使った正しいサブルーチンを示さねばなるまい。リスト 4-2 が 240~480 行と同じことをするプログラムのアセンブルリストである。以後のサンプル(リスト 4-4, 5, 6, 7, 9)では 100~170 行でこの機械語サブルーチンを使うことになる。

リスト 4-2 サブ CPU アクセスルーチン

			.Z80	
			.PHASE	0E000H
			;	
E000	C3 E006		JP	TO49
E003	C3 E01D		JP	FM49
			;	
E006	FB	TO49:	EI	
E007	EB		EX	DE,HL
E008	56		LD	D,(HL) ;A=COMMAND
E009	23		INC	HL ;INC POINTER
E00A	58		LD	E,B ;COPY COUNTER
E00B	CD E034		CALL	SEND1 ;SEND COMMAND
E00E	CD E046		CALL	CANW ;CAN SEND?
E011	F3		DI	
E012	1D		DEC	E ;DEC COUNTER
			;	
E013	56	TO49LP:	LD	D,(HL) ;GET DATA
E014	CD E034		CALL	SEND1 ;SEND 1 BYTE
E017	23		INC	HL ;INC POINTER
E018	1D		DEC	E ;DEC COUNTER
E019	20 F8		JR	NZ,TO49LP
E01B	FB		EI	
E01C	C9		RET	
			;	
E01D	FB	FM49:	EI	
E01E	EB		EX	DE,HL
E01F	56		LD	D,(HL) ;A=COMMAND
E020	23		INC	HL ;INC POINTER
E021	58		LD	E,B ;COPY COUNTER
E022	CD E034		CALL	SEND1 ;SEND COMMAND
E025	CD E046		CALL	CANW ;CAN SEND?
E028	F3		DI	
E029	1D		DEC	E ;DEC COUNTER
			;	
E02A	CD E03D	FM49LP:	CALL	GET1 ;RECEIVE DATA
E02D	72		LD	(HL),D ;STORE DATA
E02E	23		INC	HL ;INC POINTER
E02F	1D		DEC	E ;DEC COUNTER
E030	20 F8		JR	NZ,FM49LP
E032	FB		EI	
E033	C9		RET	



E034	CD	E046	;	SEND1:	CALL	CANW	;SUB-CPU READY?
E037	01	1900			LD	BC,1900H	
E03A	ED	51			OUT	(C),D	;SEND DATA
E03C	C9				RET		
E03D	CD	E050	;	GET1:	CALL	CANR	;SUB-CPU READY?
E040	01	1900			LD	BC,1900H	
E043	ED	50			IN	D,(C)	;GET DATA
E045	C9				RET		
E046	01	1A01	;	CANW:	LD	BC,1A01H	
E049	ED	78		CANWLP:	IN	A,(C)	
E04B	E6	40			AND	40H	
E04D	20	FA			JR	NZ,CANWLP	
E04F	C9				RET		
E050	01	1A01	;	CANR:	LD	BC,1A01H	
E053	ED	78		CANRLP:	IN	A,(C)	
E055	E6	20			AND	20H	
E057	20	FA			JR	NZ,CANRLP	
E059	C9				RET		
			;				
					END		

リスト 4-3 正しいサブ CPU の使い方

```

100 CLEAR&HDFFF
110 DEFUSR0=&HE000:DEFUSR1=&HE003
120 MEM$(&HE000,16)=HEXCHR$("C3 06 E0 C3 1D E0 FB EB 56 23 58 CD 34 E0 CD 46")
130 MEM$(&HE010,16)=HEXCHR$("E0 F3 1D 56 CD 34 E0 23 1D 20 F8 FB C9 FB EB 56")
140 MEM$(&HE020,16)=HEXCHR$("23 58 CD 34 E0 CD 46 E0 F3 1D CD 3D E0 72 23 1D")
150 MEM$(&HE030,16)=HEXCHR$("20 F8 FB C9 CD 46 E0 01 00 19 ED 51 C9 CD 50 E0")
160 MEM$(&HE040,16)=HEXCHR$("01 00 19 ED 50 C9 01 01 1A ED 78 E6 40 20 FA C9")
170 MEM$(&HE050,16)=HEXCHR$("01 01 1A ED 78 E6 20 20 FA C9")
180 '
190 DATES="99/02/23"
200 D$=CHR$(&HED)+"...":'GET DATES
210 DUMMYS=USR1(D$)
220 GOSUB"DISPHEX":PRINT
230 '
240 D$=CHR$(&HEC,&H85,&H93,&H18):'DATES="85/09/18" and WED
250 DUMMYS=USR0(D$)
260 END
270 '
280 LABEL"DISPHEX"
290 FORI=2TOLN(D$)
300 D=ASC(MID$(D$,I,1)):PRINT HEX$(D),
310 NEXT:RETURN

```

ここで、当然のことのようにリスト 4-2 を解説するのであった。使い方はリスト 4-3 を見れば分かると思うが、念のために言うと、

E000<sub>H</sub>がタイプ 1 (全部 Z80 → 80C49)

E003<sub>H</sub>がタイプ 2 (最初の一つ以外は 80C49 → Z80)

のそれぞれエントリーである。DE レジスタにコマンドとデータ部分へのポインタを、B レジスタにその両方を合わせたバイト数を入れて CALL すればよい。これは BASIC で USRn(Δ\$) を実行したときにセットされるレジスタの内容に合わせているので、リスト 4-3 のように手軽に使えるのである。タイプ 1、タイプ 2 両方とも処理の流れは同じようなものであるから、タイプ 1 についてだけ処理を追ってみる。

まず E000<sub>H</sub>でいきなり「TO49」へジャンプしてしまう。つまりそこが本当の始まりなのだ。最初に EI で割り込みを許可している。一見無意味のようだが、これには深い背景



と、私の一晩の苦しみが込められているのだ。X1では動作中ほとんどの場合において割り込みが許可（つまり EI）されているのだが、これは主にキー入力のためである。さて、前にも書いたように、そのキー入力による割り込みもサブ CPU によって行なわれているのだ。このことから分かるように、サブ CPU と（キー割り込み以外のことで）交信しようとする場合は、割り込みを禁止（DI）しておかなければならないのだ。だが、そのタイミングは凶悪そのものである。すなわち、リスト 4-2 にもあるように、最初のコマンドを送った後、サブ CPU が「またデータを送ってもいいよ」と言うまで割り込みを禁止してはいけないのだ。これはタイプ 1、タイプ 2 の両方について言えることである。私はこれを発見するのに一晩かかってしまった。最後にはとうとう仕方なく HuMonitor を解析して謎が解けたのである。この取り決めを守らないと、リスト 4-1 での **BREAK** キーの速押しなどで動作がその筋してしまうのである。この点のほかは、RET する前に EI を実行する点以外に特に説明を必要とする部分はないと思う。しかし、縁起ものだから一応言っておくが、それぞれのルーチンは、

- SEND1：80C49 から OK サインが出るまで待つて 1 バイト送る
- GET1：80C49 から OK サインが出るまで待つて 1 バイト受け取る
- CANW：80C49 へデータを送れるようになるまで待つ
- CANR：80C49 からデータを受け取れるようになるまで待つ

となっている。BASIC から USR 命令で呼び出す分にはかなり便利だろうと思う（実は BASIC の IOCS 中にはもっと便利なサブルーチンがあるのだが、それを使うのは軟弱であるという私の独断により、一切無視するのであった）。

では、具体的にそれぞれのコマンドについて解説を始める。

#### ① E3<sub>H</sub>（turbo のゲームキー読み取り）

これは turbo のキーボードのスイッチを、「B」にしたときに発動される秘密兵器である。簡単に言ってしまうと 24 個のキーを（ほとんど）同時にリアルタイムに読み出せるのである。24 個のキーは固定で、テンキーとメインキーの **S** を中心とした Q～C の 8 個、およびその周辺のキーである。具体的にどのキーかということは、**図 4-1** を見ていただきたい。

次のサンプルプログラムは**リスト 4-4**である。100～170 行はリスト 4-3 の一部分を流用したものである。230～250 行は後程説明する。このプログラムでは BIN\$を使って、押されたキーに対応するビットを 1 にして表示する。ループにしてあるのでリアルタイムに変化する様子をながめていただきたい。充分味わったなら、キーボードのスイッチを「A」にしていいただきたい。何を押しても全部 0 のままのはずである。すなわちキーボードがモード A のときは何を押しても 0 が返ってくるのだ。

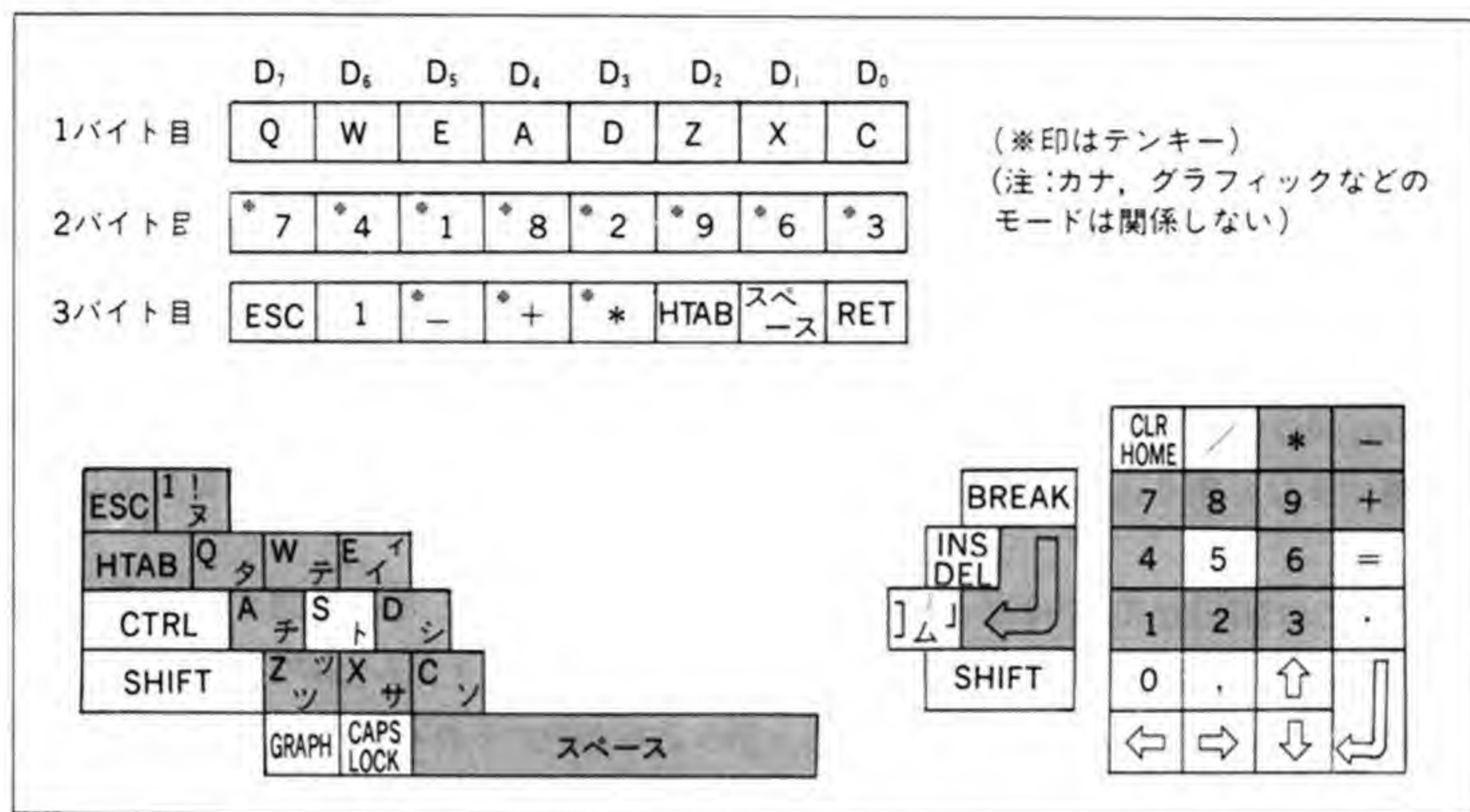
#### ② E4<sub>H</sub>（キー入力割り込みベクトルセット）

出たっ！ とうとう割り込みである。この割り込みというやつはなかなか面倒臭いので心していただきたい。

まずは一般論である。割り込みというのは、一心不乱に仕事をしている人に電話がかかってくるようなものである。その人は電話のベルを聞くと、今どの仕事のどの部分をやって



図 4-1 ゲームキー配置図



リスト 4-4 キー入力

```

100 CLEAR&HDFFF
110 DEFUSR0=&HE000:DEFUSR1=&HE003
120 MEM$(&HE000,16)=HEXCHR$("C3 06 E0 C3 1D E0 FB EB 56 23 58 CD 34 E0 CD 46")
130 MEM$(&HE010,16)=HEXCHR$("E0 F3 1D 56 CD 34 E0 23 1D 20 F8 FB C9 FB EB 56")
140 MEM$(&HE020,16)=HEXCHR$("23 58 CD 34 E0 CD 46 E0 F3 1D CD 3D E0 72 23 1D")
150 MEM$(&HE030,16)=HEXCHR$("20 F8 FB C9 CD 46 E0 01 00 19 ED 51 C9 CD 50 E0")
160 MEM$(&HE040,16)=HEXCHR$("01 00 19 ED 50 C9 01 01 1A ED 78 E6 40 20 FA C9")
170 MEM$(&HE050,10)=HEXCHR$("01 01 1A ED 78 E6 20 20 FA C9")
180 '
190 D$=CHR$(&HE3)+"...":'READ GAME KEY
200 DUMMY$=USR1(D$)
210 GOSUB"DISPBIN"
220 '
230 D$=CHR$(&HE6)+"...":'READ NORMAL KEY
240 DUMMY$=USR1(D$)
250 GOSUB"DISPHEX":PRINT
260 GOTO190
270 END
280 '
290 LABEL"DISPBIN"
300 FORI=2TOLEN(D$)
310   D=ASC(MID$(D$,I,1)):PRINT RIGHT$("0000000"+BIN$(D),8),
320 NEXT
330 RETURN
340 '
350 LABEL"DISPHEX"
360 FORI=2TOLEN(D$)
370   D=ASC(MID$(D$,I,1)):PRINT HEX$(D),
380 NEXT
390 RETURN

```

いたかを覚えておいて、電話を取るわけである。なぜ覚えておくかというと、電話が済んだ後で元の仕事に遅滞なく戻るためである。コンピュータの場合も同じようなもので、CPUは「それフェッチしてきたぞ。なにになに3E<sub>H</sub>だと、よっしゃ、もう1バイト持ってきてAレジスタにおち込めばよいのだな。そうれっ！次は何だ？おっとC3<sub>H</sub>だぜ。今日はジャンプが多いなあ」などつつぶやきつつ、一心不乱に仕事をしているのである（たぶん）。このCPUに電話、じゃない、割り込みをかけるには、CPUからムカデ状に出ている足に電圧をかけてやればよいのである。Z80の場合は割り込み用の足を2本持っている。16



番ピンと 17 番ピンで、おのこの  $\overline{\text{INT}}$ 、 $\overline{\text{NMI}}$  と呼ばれている。上に横棒（こんなやつ）が付いているのは、「0 ボルトになると効きまーす」という意味なのだ。だから、このピンを 0 ボルト（つまり Low）にすると、CPU に割り込みがかかるのである。 $\overline{\text{NMI}}$  の方は例のリセットキーである (IPL じゃない方)。こいつは実に我儘で、Z80 はこの割り込みに対して拒否権がないのである。すなわち、

#### Non Maskable Interrupt

なのだ。Mask には「邪魔をする」という意味がある。Maskable で「邪魔できる」。それに Non がついて「邪魔できない」。最後の Interrupt は「割り込み」そのものだから、結局「邪魔できない割り込み」ということになる。うーん、いきなり英文読解をしてしまった。

実はこの  $\overline{\text{NMI}}$  は X1 ではほとんど意味がないので、暇なときに勉強すれば充分である。もっと大事なのが  $\overline{\text{INT}}$  で、これがキー入力割り込みベクトルに関係してくる。

さて、この  $\overline{\text{INT}}$  の足が Low にされると割り込みがかけられるわけであるが、こっちに対してなら Z80 は拒否権を持っている。先程も出てきた DI である (電話の受話器を外しておくようなもの)。Z80 がこの命令を実行すると、「わしゃ、 $\overline{\text{INT}}$  がどーなろうとも知らんもんね」を決め込むのである。これの逆が EI で、これを実行すると、「私はどんな割り込みでも受けるっ！」となる。ただし正確には EI の次の命令の実行後である。つまり、

EI

RET

という命令が並んでいると、RET 命令でサブルーチンからリターンした後で割り込みを受け付けるようになるのである。

それはさておき、こっちの割り込みには 3 種類があり、Z80 が自由に選択できるのだ。それぞれモード 0、モード 1、モード 2 と呼ばれる (具体的には「IM 0」、「IM 1」、「IM 2」という見慣れない命令による)。このうちモード 0 とモード 1 は 8080 コンパチなモードである。そして実にセコイのだ。当然のごとく X1 ではそんなものは使わないのである。よって、これらの二つのモードについても暇なときに勉強すればよい。というところで、 $\overline{\text{INT}}$  のモード 2 の、割り込みにやっとたどりついた私であった。

知る人ぞ知る隠れレジスタというのが Z80 にある。F レジスタと I レジスタと R レジスタである。このうち I レジスタ (インタラプトレジスタ) というのがモード 2 の割り込みでは主役の一人である。これは 8 ビットのレジスタである。

モード 2 の割り込みが発生すると、Z80 はおもむろにこの I レジスタを見るのだ。次に割り込み源 (誰だっ！俺の  $\overline{\text{INT}}$  を Low にしたやつは?) から 1 バイトのデータを受け取るのだ (これを割り込みベクトルもしくは割り込みベクタという)。次に Z80 は I レジスタを上位、割り込みベクトルを下位とする 2 バイトの数字を作り出し、それをアドレスとみなして、その番地に何が書いてあるかを見るのである (見るだけ)。そして Z80 は **そこに書いてあるアドレスへ** サブルーチンコールを起こすのである (正確にはサブルーチンコールではないが、ほとんど同じ)。X1 の BASIC (turbo は違う) では I レジスタが 00<sub>h</sub> で、キー入力による割り込みベクトルが 52<sub>h</sub> である。そこで Z80 は 0052<sub>h</sub> 番地からの 2 バイ

86 試験に出る X1



トに格納されているアドレスに飛んでいくのである。つまり、X1のBASICでは、最終的にZ80は、

$\text{PEEK}(\&\text{H0052}) + \text{PEEK}(\&\text{H0052}) * 256 = \&\text{H0346}$

に制御を移すのである。ここでZ80の声を聞いてみるのである。

**Z80:**「おっ！誰でい、誰でい、おいらの $\overline{\text{INT}}$ をLowにするやつは。えーと、今の割り込みモードは……ああ、モード2になつてら。それに割り込みも許可してあらーな。よしよし、分かったから割り込みベクトルをよこしな。ふうん、52<sub>H</sub>ねえ。まあいいだろう。さてと、Iレジスタの方は……00<sub>H</sub>ねえ。となると、ちょっと0052<sub>H</sub>番地からの2バイトをのぞいてみるとすっか。えーとなになになに、0346<sub>H</sub>と書いてあるな。そんじゃちょっと一仕事とくらあ。あらよっ」

となるのであった。

それで結論である。つまりこのE4<sub>H</sub>というコマンドは、そのキー入力用の割り込みベクトルを設定するものなのだ。今書いたように、X1のBASIC(CZ-8CB01, FB01のVer 1.0, 2.0の四つ)ではIレジスタが00<sub>H</sub>、キー入力の割り込みベクトルが52<sub>H</sub>である。turbo BASICではIレジスタがF8<sub>H</sub>、キー入力の割り込みベクトルが1A<sub>H</sub>である。

そこでリスト4-5のサンプルプログラムを見ていただきたい。これはCB01, FB01用で、turbo BASICでは動かない。このプログラムは、キー入力ベクトルを変えて、0052<sub>H</sub>ではなく0054<sub>H</sub>番地を参照するようにしている。そこにはE060<sub>H</sub>を書き込んであるので、結局何かキーを押すと、Z80はE060<sub>H</sub>に飛んでくることになる。E060<sub>H</sub>で手ぐすね引いて待っているのは200, 210行にある機械語プログラムである。最後にある「00 30」は変数エリアで、このI/Oアドレスの所へ65<sub>H</sub>(200行の中程にある)をOUTし、変数を一つ増やして(たとえば3000<sub>H</sub>→3001<sub>H</sub>)から本当のキー入力ルーチンの3046<sub>H</sub>番地に飛んでいる。要するにこのプログラムを1回走らせて「OK」が出ると、それ以後何かキーを押すたびに画面に「e」が1個ずつ現れるのだ。正確には、キーを離したときにもキー割り込みが起きるので、ポンと1回キーを押すと「ee」と現れる。変数エリアにある3000<sub>H</sub>はそのたびに増加するので、しばらく続けるとグラフィックRAMに65<sub>H</sub>を書き込み始めることになり、青い点線が画面をトコトコ走るであろう。書き込みアドレスを「00 30」ではなく、最初から「00 40」にしておくという手もある。打ち間違いがあると当然キー入力ができなく

#### リスト4-5 キー割り込みにオジャマ

```
100 CLEAR&HFFFF
110 DEFUSR0=&HE000:DEFUSR1=&HE003
120 MEM$(&HE000,16)=HEXCHR$("C3 06 E0 C3 1D E0 FB EB 56 23 58 CD 34 E0 CD 46")
130 MEM$(&HE010,16)=HEXCHR$("E0 F3 1D 56 CD 34 E0 23 1D 20 F8 FB C9 FB EB 56")
140 MEM$(&HE020,16)=HEXCHR$("23 58 CD 34 E0 CD 46 E0 F3 1D CD 3D E0 72 23 1D")
150 MEM$(&HE030,16)=HEXCHR$("20 F8 FB C9 CD 46 E0 01 00 19 ED 51 C9 CD 50 E0")
160 MEM$(&HE040,16)=HEXCHR$("01 00 19 ED 50 C9 01 01 1A ED 78 E6 40 20 FA C9")
170 MEM$(&HE050,10)=HEXCHR$("01 01 1A ED 78 E6 20 20 FA C9")
180 '
190 POKE&H54,&H60,&HE0
200 MEM$(&HE060,16)=HEXCHR$("F5 C5 ED 4B 74 E0 3E 65 ED 79 03 ED 43 74 E0 C1")
210 MEM$(&HE070,6)=HEXCHR$("F1 C3 46 03 00 30")
220 '
230 INIT
240 D$=CHR$(&HE4,&H54):'SET KEY VECT.
250 DUMMY$=USR0(D$)
```

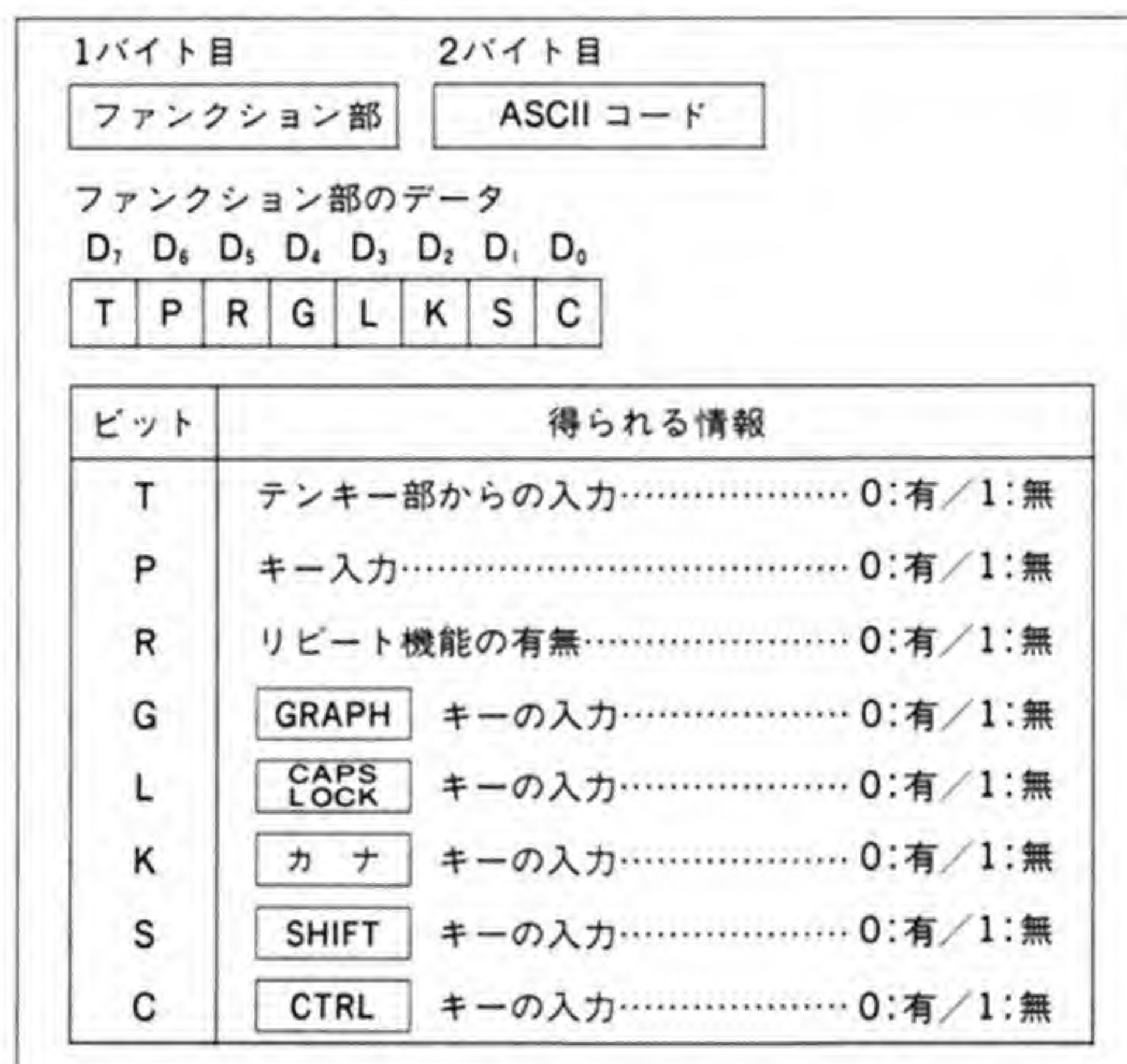


なったりするので、リセットスイッチの助けを借りること。

### ③ E6<sub>H</sub> (キーデータの読み出し)

X1 では普通キー入力には割り込みを使うわけだが、別にその方法に限定されているわけではないのである。割り込みを使わずにキーを読むには、サブ CPU に E6<sub>H</sub>を送った後2バイトのデータを受け取ればよい。データの内容は図 4-2 に示すように、最初の1バイトがファンクション部と呼ばれるもので、早い話が BASIC の INKEY\$(2)と同じである。次に受け取る1バイトが ASCII コードそのものである。リスト 4-4 の 230～250 行がサンプルである。このプログラムは割り込みと共存しており、**BREAK** キーを押すと BASIC が一瞬停止するので、その状態を見ることはできない。**SHIFT** + **BREAK** も同様である。

図 4-2 キーボードからのデータの内容



さて、実はこのファンクション部は X1 と turbo では少々違うのである。X1 では有効なキーが押されていないければ、ファンクション部は常に FF<sub>H</sub> (ビットが全部 1) なのだが、turbo ではビット 4～0 (図 4-2 の G～C) の 5 ビットが有効なキーが押されていないくとも ON/OFF するようになっている。たとえば turbo で **SHIFT** キーだけを押ししても、それを検知できるのだ。確認する方法は簡単で、

```
10 PRINT ASC(INKEY$(2))
20 GOTO 10
```

を RUN すればよい。キーボードのモードが A, B いずれでも同じである。この部分は(おそらく唯一の) 非コンパチ部分であろう。モード A のときは X1 と同じになるようになっていれば天晴だったと思うが、実害はないようなので許してしまうのである。



ところで、当然割り込みを使った場合も示す必要があるが、ちょっと様相が異なるので最後に回すことにする。

#### ④ E7<sub>H</sub>, E8<sub>H</sub> (TV コントロール/読み出し)

E7<sub>H</sub>は大体 BASIC の CRT, TVPW, CHANNEL, VOL コマンドに対応する。しかし BASIC の命令ではサポートされていない機能のチャンネルコール, チャンネルの順/逆送りなども実行できるのである。E7<sub>H</sub>の次のデータが行なうコントロールを図 4-3 に示す。サンプルプログラムはリスト 4-6 である。これは入力されたデータを 16 進数とみなして E7<sub>H</sub>に続けてサブ CPU に送るものである。事前に CRT のスイッチを切ってから「 $\triangleleft$ 」を押すなどして楽しんでいただきたい。このプログラムでは,「 $\cdot$ 」を入力すると E8<sub>H</sub>コマンドを使ってサブ CPU からデータを受け取り,表示するようになっている。まかり間違っても「E8<sub>H</sub>コマンドは最後に実行された E7<sub>H</sub>コマンドで送られたデータを返すだけ」などとは思わな

図 4-3 TV コントロールの内容

E7 <sub>H</sub> +	01 <sub>H</sub>	音量アップ
	02 <sub>H</sub>	音量ダウン
	03 <sub>H</sub>	音量ノーマル
	04 <sub>H</sub>	チャンネルコール(トグル)(turboのディスプレイCZ-850Dなどのみ)
	05 <sub>H</sub>	TV画面
	06 <sub>H</sub>	音量ミュート(トグル)
	07 <sub>H</sub>	不明。なぜかチャンネルが4になる
	08 <sub>H</sub>	TV/COM画面(トグル)
	09 <sub>H</sub>	チャンネルコール(トグル)(turboのディスプレイCZ-850Dなどは受け付けない)
	0A <sub>H</sub>	スーパーインポーズコントラストノーマル(一気型)
	0B <sub>H</sub>	チャンネル順送り
	0C <sub>H</sub>	チャンネル逆送り
	0D <sub>H</sub>	パワーオフ
	0E <sub>H</sub>	パワーオン/オフ(トグル)
	0F <sub>H</sub>	スーパーインポーズコントラストダウン(一気型)
	10 <sub>H</sub>	チャンネル1
	11 <sub>H</sub>	チャンネル2~11
	1B <sub>H</sub>	チャンネル12
	1C <sub>H</sub>	TV画面
	1D <sub>H</sub>	COM画面
	1E <sub>H</sub>	スーパーインポーズコントラストダウン
	1F <sub>H</sub>	スーパーインポーズコントラストノーマル
	80 <sub>H</sub>	TVパワーオン

- 注●0A<sub>H</sub>と1F<sub>H</sub>, 0F<sub>H</sub>と1E<sub>H</sub>は基本的に同じ動作だが、一気型の方が素早くスーパーインポーズになる。  
●05<sub>H</sub>と1C<sub>H</sub>の違いは不明。  
●01<sub>H</sub>~1F<sub>H</sub>のデータに80<sub>H</sub>を加えたものは、TVパワーオン後にそれぞれの動作をする。たとえば86<sub>H</sub>=80<sub>H</sub>+06<sub>H</sub>は、TVパワーオン後に音声ミュートする。



#### リスト 4-6 TV コントロール

```

100 CLEAR&HDFFF
110 DEFUSR0=&HE000:DEFUSR1=&HE003
120 MEM$(&HE000,16)=HEXCHR$("C3 06 E0 C3 1D E0 FB EB 56 23 58 CD 34 E0 CD 46")
130 MEM$(&HE010,16)=HEXCHR$("E0 F3 1D 56 CD 34 E0 23 1D 20 F8 FB C9 FB EB 56")
140 MEM$(&HE020,16)=HEXCHR$("23 58 CD 34 E0 CD 46 E0 F3 1D CD 3D E0 72 23 1D")
150 MEM$(&HE030,16)=HEXCHR$("20 F8 FB C9 CD 46 E0 01 00 19 ED 51 C9 CD 50 E0")
160 MEM$(&HE040,16)=HEXCHR$("01 00 19 ED 50 C9 01 01 1A ED 78 E6 40 20 FA C9")
170 MEM$(&HE050,10)=HEXCHR$("01 01 1A ED 78 E6 20 20 FA C9")
180 '
190 INPUT A$:IF A$="." GOTO 240
200 CODE=VAL("&H"+A$)
210 D$=CHR$(&HE7,CODE):DUMMY$=USR0(D$):BEEP
220 GOTO190
230 '
240 D$=CHR$(&HE8)+". "
250 DUMMY$=USR1(D$)
260 GOSUB"DISPHEX":PRINT
270 GOTO 190
280 '
290 LABEL"DISPHEX"
300 FORI=2TOLEN(D$)
310 D=ASC(MID$(D$,I,1)):PRINT HEX$(D),
320 NEXT:RETURN

```

のように。E8<sub>H</sub> コマンドは、あくまでサブ CPU から TV に最後に送られたデータを返すのである。もちろん E7<sub>H</sub> で送られたものもこのうちに入るが、キーボードを使って TV をコントロールした場合（たとえば **SHIFT** + **1**）も TV に送られたデータになるのだ（この場合は 10<sub>H</sub> になる）。さらに TV タイマの実行時間がきて TV コントロールが起こった場合も同様である。この点を心得ておくように。

#### ⑤ E9<sub>H</sub>, EA<sub>H</sub> (カセットデッキコントロール/状態読み出し)

E9<sub>H</sub>, EA<sub>H</sub> は実に単純明解なことに BASIC の CMT 命令と同じである。マイナーな命令だから知らない人もいるだろうから、そのよーな場合はマニュアルを読んでいただきたい。すなわち、

E9<sub>H</sub> は CMT = × ×

EA<sub>H</sub> は ○ ○ = CMT

に対応する。

早い話がこれだけのことなのだが、偶然にも私は X1D の BASIC のマニュアルでは CMT コマンドの説明が違ふことを発見してしまった。すなわち X1D では外部テープデッキのリモートの ON/OFF ということになっているのだ。

#### ⑥ EB<sub>H</sub> (カセットセンサー読み出し)

これは BASIC の CMT (△) 関数に対応している。図 4-4 と BASIC のマニュアルを見比べていただきたい。なお、例によって X1D の BASIC マニュアルにはこれに対応するものはない。

#### ⑦ EC<sub>H</sub>, ED<sub>H</sub> (カレンダーセット/読み出し)

BASIC の DATE\$, DAY\$ に相当するものである。データの内容は図 4-5 に示すような形式になっている。

図にあるように、1 バイト (8 ビット) を上下 4 ビットずつに分けて、それぞれの 4 ビットで 0 ~ 9 までの数値 (だけ) を指定し、結局 00 ~ 99 (10 進数) を表現する方法を BCD (2 進化 10 進数) という。なにやら面倒臭いが、早い話 4 ビットなら 0 ~ 15 までの数値を



図 4-4 カセットセンスデータの内容

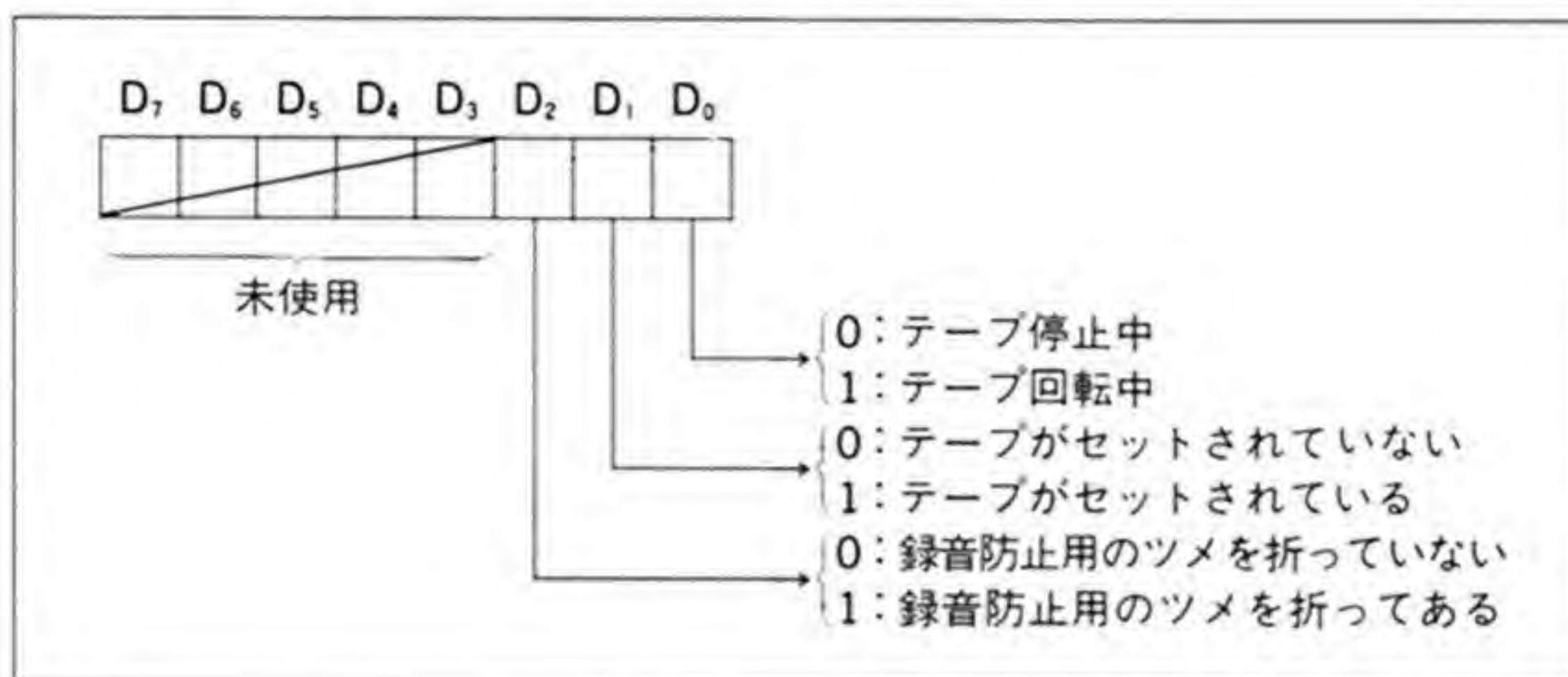
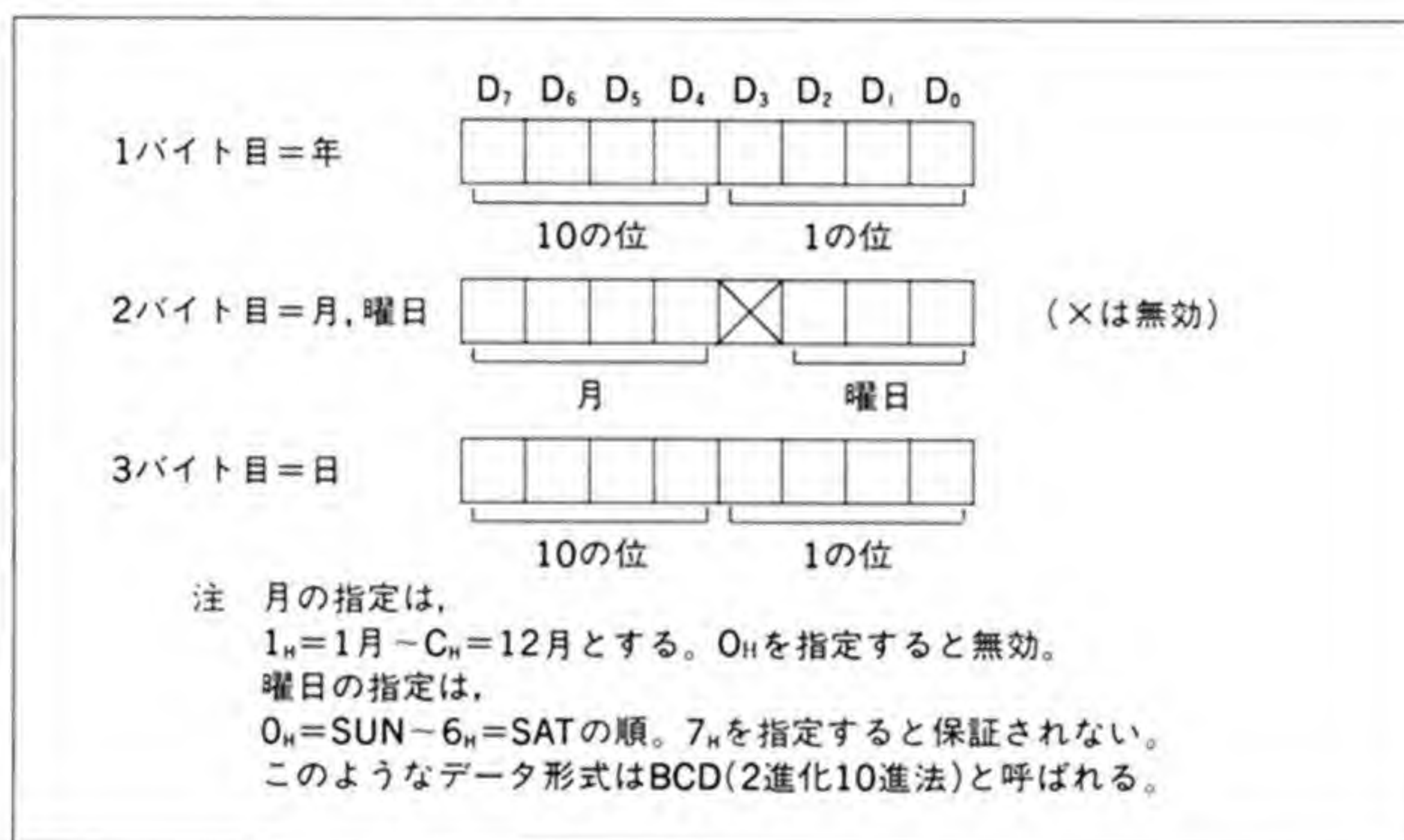


図 4-5 カレンダーデータの内容



表現できるところを0～9に制限して、その代わりに（ある意味で）扱いやすくしようというものである。この場合において最大の利点といえば、サンプル（リスト4-3）にもあるように「&H」や「HEX\$」を使い16進数のつもりで10進数を扱える点である。以上の説明でも理解できないなら、これでどーだ。

### 「3バイトがあるのではなく、4ビットが6組あると思え」

さて、すぐ思いつくのが、「17月48日」などのめっちゃくちゃな日付を設定するとどうなるかである。結果は日付がめっちゃめっちゃになるだけで、特に面白いことは起こらないようである。夜中の12時を過ぎると、存在しうる日付に訂正されてしまう。なお、大晦日を過ぎても「年」は増加しない。また、言語道断の悪習、閏年はサポートされていない。

最後に一言注意。日付などは、夜中の12時になると、その瞬間に更新されるのではなく、数秒の遅れがあるようである。これを避けるためには、日付を読み出す直前に時計の読み出しをすればよい。そうすれば自動的に日付も時計が示すとおり更新されるようである。

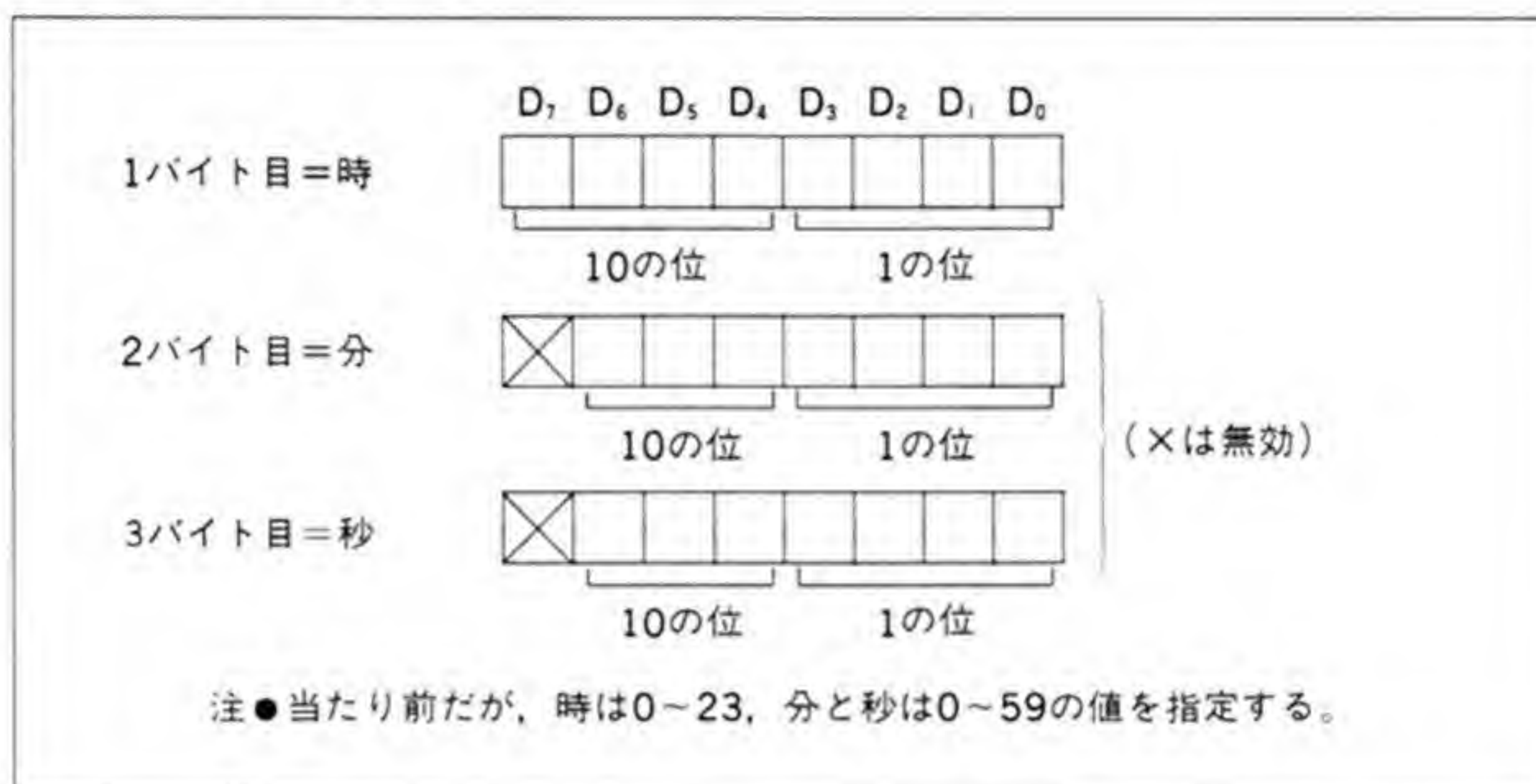


# ⑧ EE<sub>H</sub>, EF<sub>H</sub> (時計セット/読み出し)

BASIC の TIMES に相当するものである。形式は図 4-6 に示すとおり。

時, 分, 秒の数値を範囲を超えて設定しても, たちまち正常な値に訂正されてしまうようである。

図 4-6 時計データの内容



# ⑨ D0<sub>H</sub>~D7<sub>H</sub>, D8<sub>H</sub>~DF<sub>H</sub> (タイマセット/読み出し)

いよいよサブ CPU のコマンドも最後である。しかしここで気を抜いてはいけない。なぜかという、この TV タイマは隠れ機能の宝庫だからなのだ。そのへんは参考文献 8 が詳しいようである。

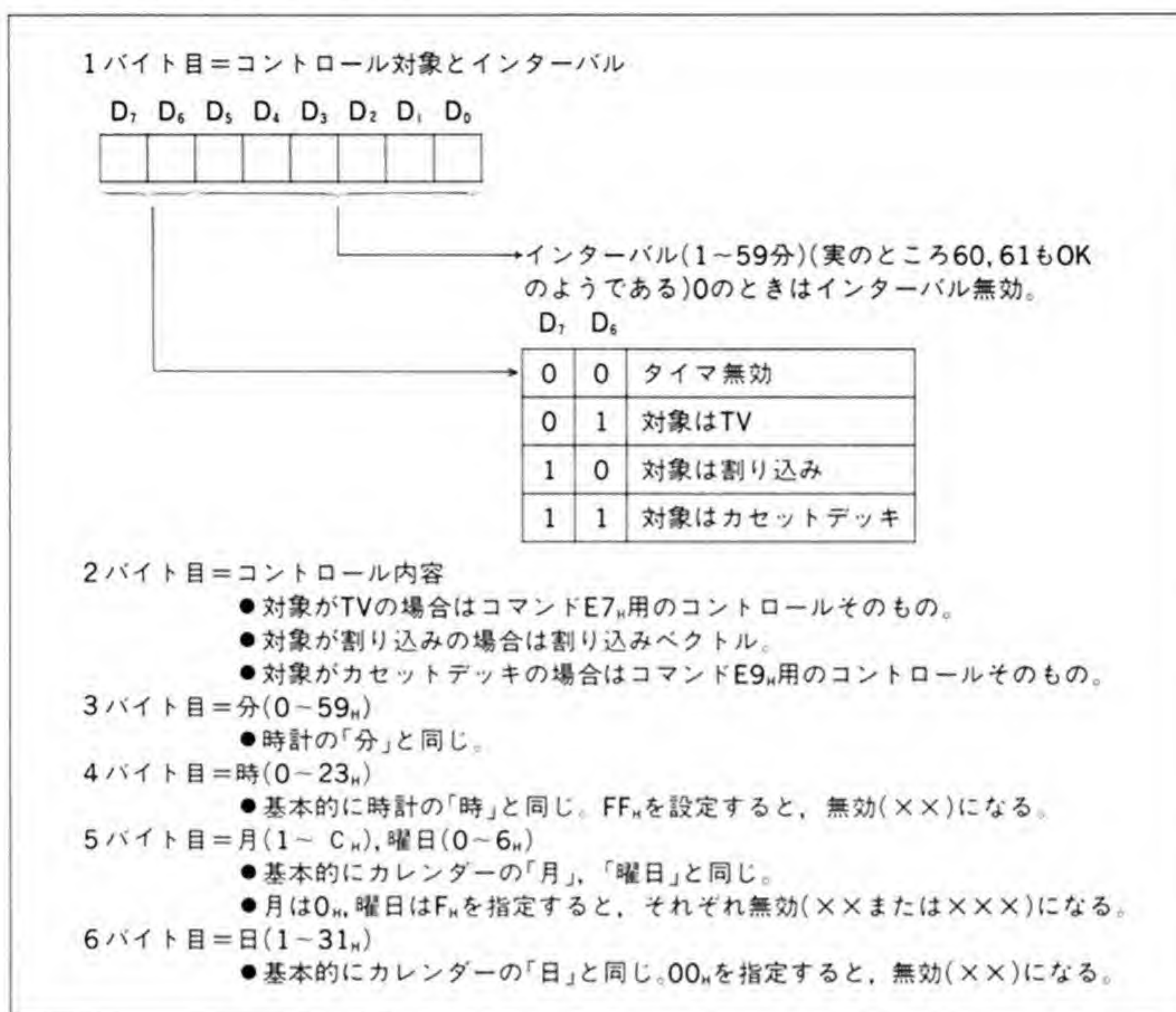
まず図 4-7 を見ていただきたい。いきなり出てくる「対象は割り込み」である。そうなのである。このタイマは TV のコントロールだけではなく、割り込みやカセットデッキのコントロールもやってしまうのである。しかも、何に使ってよいやら理解に苦しむインターバル機能まである。では、おもむろに始めよう。

気付いた方もいるだろうが、実は X1 のタイマは全部で 8 個あるのだ。ところがぎゅっちゃん、ASK コマンドなどでタイマ設定モードに入ってもタイマは 7 個しか表示されない。すなわち、「タイマ 0」は隠れタイマなのであった。しかも ASK で設定できるのは TV コントロールのみで、さらに挙げ句の果てにはそのコントロールも 90<sub>H</sub>~9B<sub>H</sub>および 0D<sub>H</sub> (パワー OFF) だけなのである。インターバルというのは、タイマに設定された時刻がきたらまず普通に (TV OFF などを) 実行し、その後はインターバルに指定された「分」ごとに同じ動作を繰り返すというものである。たとえばインターバルを 1 分に指定し、動作を音声ミュートにすると、この動作はトグルだから、1 分ごとに TV の音が出たり出なくなったりするわけである。いたずらして友達を驚かせようとしているのはあなただけではない。

次に D<sub>7</sub>, D<sub>6</sub>を見ると、「10」で対象が割り込みとなる。すなわちこの場合は、タイマに設定した時刻になるとサブ CPU から割り込みがかけられるのである。もちろんインターバルも有効である。このおいしそうな部分は後程ゆっくりと料理する。D<sub>7</sub>, D<sub>6</sub>が「11」で対象がカセットテープになる。つまり、設定された時刻になるといきなりテープが回り出すわけである。コントロール内容は E9<sub>H</sub>で使えるものがすべて OK だから、突然録音状態に突



図 4-7 タイマ設定/読み出しの内容



入するなどという凶悪な事態も存在し得るのである。おーこわ。

そのよーな状態であるから、もし万が一、暴走によって

### タイマが異常にセットされた

ならば、ゲームがいきなり止まってしまう、などということがあり得る。そのよーな症状がある場合には次のようなおはらいをしていただきたい。

- 1) 本体裏面にあるメイン電源スイッチを OFF にする。
- 2) ポンポンと2回拍手を打つ。
- 3) メイン電源スイッチを ON にする。

以上でサブ CPU に設定されたタイマはすべてクリアされる。

先程も書いたように、タイマ0はASK コマンドによるタイマ設定モードにおいても表示されない隠れタイマであるから、そのよーな意味も含めて1か月に一度ぐらいは以上のようなおはらいを励行してもよいのではないかと思う私であった(タイマLEDがついていないのにやるのもナンだけど)。

というところで、サンプルプログラムはリスト 4-7 である。やっていることは、まず 230 行で日付と時刻を9月18日の16時10分55秒にしている。タイマ1は、16時11分に設定されており、動作は08<sub>H</sub>のTV/COM画面トグルである。タイマ2は割り込みで、「&H80+



# リスト 4-7 タイマの使い方 3 態

```

100 CLEAR&HDFFF
110 DEFUSR0=&HE000:DEFUSR1=&HE003
120 MEM$(&HE000,16)=HEXCHR$("C3 06 E0 C3 1D E0 FB EB 56 23 58 CD 34 E0 CD 46")
130 MEM$(&HE010,16)=HEXCHR$("E0 F3 1D 56 CD 34 E0 23 1D 20 F8 FB C9 FB EB 56")
140 MEM$(&HE020,16)=HEXCHR$("23 58 CD 34 E0 CD 46 E0 F3 1D CD 3D E0 72 23 1D")
150 MEM$(&HE030,16)=HEXCHR$("20 F8 FB C9 CD 46 E0 01 00 19 ED 51 C9 CD 50 E0")
160 MEM$(&HE040,16)=HEXCHR$("01 00 19 ED 50 C9 01 01 1A ED 78 E6 40 20 FA C9")
170 MEM$(&HE050,10)=HEXCHR$("01 01 1A ED 78 E6 20 20 FA C9")
180 '
190 POKE &H54,&H60,&HE0:'SET INT. TABLE
200 MEM$(&HE060,10)=HEXCHR$("F5 3E 01 32 70 E0 F1 FB ED 4D")
210 POKE &HE070,0:'INIT INT. FLAG
220 CLS4:WIDTH80
230 DATE$="85/09/18":TIME$="16:10:55"
240 '
250 D$=CHR$(&HD1,&H40,&H8,&H11,&H16,&H9F,&H18):'COM. SCR. ON 16:11:00
260 DUMMY$=USR0(D$)
270 D$=CHR$(&HD2,&H80+1,&H54,&H12,&H16,&H9F,&H18):'INTERRUPT ON 16:12:00
280 DUMMY$=USR0(D$)
290 D$=CHR$(&HD3,&HC0,&H0,&H13,&H16,&H9F,&H18):'EJECT ON 16:13:00
300 DUMMY$=USR0(D$)
310 '
320 D$=CHR$(&HD8+4)+". . . . .":'READ TIMER
330 DUMMY$=USR1(D$)
340 GOSUB"DISPHEX":PRINT
350 '
360 ADR=&HE070:KAISU=0
370 LOCATE 0,1:PRINT TIME$
380 IF PEEK(ADR) THEN POKE ADR,0:KAISU=KAISU+1:'CHECK INT.
390 PRINT KAISU:'ワソコミ ノ カイスウ
400 GOTO 370
410 END
420 '
430 LABEL"DISPHEX"
440 FORI=2TOLLEN(D$)
450 D=ASC(MID$(D$,I,1)):PRINT HEX$(D),
460 NEXT:RETURN

```

1」となっていることから分かるように、1分のインターバルを指定している。すなわち、16時12分以降1分ごとに割り込みがかけられるわけである。タイマ3はカセットの制御で、16時13分にEJECTが行なわれる。TVコントロールとカセットの制御は見れば分かるだろうから、タイマによる割り込みを少し説明する。割り込みベクトルは54<sub>H</sub>である。すなわち割り込みが発生したら0054<sub>H</sub>番地に書かれているアドレスへ飛ぶ。そこで190行では、その飛び先であるE060<sub>H</sub>を書き込んである。実際に割り込みを処理するルーチンは200行にある。実に簡単なプログラムで、

```

PUSH AF
LD A, 1
LD (0E070H), A
POP AF
EI
RETI

```

となっている。これにより割り込みがかけられるとE070<sub>H</sub>番地に1がセットされる。そこで割り込みが発生したかどうかをチェックするBASICプログラムが360~400行の無限ループである。ずっとE070<sub>H</sub>番地の内容をPEEKを使って見張っていて、1がセットされたら検知するようにしている。

さて、正しいその筋の読者はここであるアイデアを思いつかなければならない。つまり、「16時12分に割り込みがかかったなら、その割り込み処理ルーチンの中で、時計を16



時 11 分 59 秒に設定すれば、その 1 秒後にまた割り込みがかかるに違いない。時計は狂ってしまうが、分単位ではなく秒単位で割り込みをかけることができる！」。

ところが残念ながら、そうはいかない。時計をしょっちゅう読み出し続けていれば（たとえば、PRINT TIMES\$の繰り返し）2 秒ごとの割り込みなども可能なようだが、さもなくば 1 分未満の間隔での割り込みは（まともには）無理のようである。

さらに複数のタイマが同時に割り込みを起こす場合もあり得るが、そのような状況に対してはなんらかの優先順位に基づいて、ちゃんとそれぞれ割り込みを起こすようである。

最後になったが、リスト 4-8、4-9 である。リスト 4-8 はキー割り込みの処理ルーチンのアセンブルリストである。見て分かるように、実はこの場合はキーデータを読み出すのに際して、サブ CPU にコマンド E6<sub>H</sub>を送っても送らなくてもよいのだ。また、turbo では RETI する前に、サブ CPU に E3<sub>H</sub>を送ってやればゲームキーも読める。リスト 4-9 がこのルーチンを使ったサンプルである。割り込みベクトル 54<sub>H</sub>を使い、E060<sub>H</sub>～のルーチンでキー入力を処理している。ESC を押すとベクトルを 52<sub>H</sub>にして終わる。BASIC では、バッファを持っていて、キー入力をためておくことができるのだが、このルーチンではそんなことはせずに、E080<sub>H</sub>番地からの 2 バイトにファンクション部と ASCII 部のデータを書き込んでいるだけである。

てなわけであった。いざ自力でオールマシン語のゲームでも作ろうとすると、サブ CPU は案外、避けて通れぬドラゴンだったりする。この章を噛みしめて、力一杯その筋していただきたい。

リスト 4-8 キー割り込み処理ルーチン

			.280
			.PHASE 0E060H
		;	
E034		SEND1	EQU 0E034H
E03D		GET1	EQU 0E03DH
E080		KEYSTR	EQU 0E080H
		;	
E060	F5	INTK:	PUSH AF
E061	C5		PUSH BC
E062	D5		PUSH DE
E063	E5		PUSH HL
		;	
		;	LD A,0E6H ;D=COMMAND
		;	CALL SEND1 ;SEND COMMAND
		;	
E064	1E 02		LD E,2
E066	21 E080		LD HL,KEYSTR
E069	CD E03D	DATA2:	CALL GET1 ;RECEIVE DATA
E06C	72		LD (HL),D ;STORE DATA
E06D	23		INC HL ;INC POINTER
E06E	1D		DEC E ;DEC COUNTER
E06F	20 F8		JR NZ,DATA2
		;	
E071	E1		POP HL
E072	D1		POP DE
E073	C1		POP BC
E074	F1		POP AF
E075	FB		EI
E076	ED 4D		RETI
		;	
			END



リスト 4-9 BASIC から「リスト 4-8」を使う

```

100 CLEAR&HDFFF
110 DEFUSR0=&HE000:DEFUSR1=&HE003
120 MEM$(&HE000,16)=HEXCHR$("C3 06 E0 C3 1D E0 FB EB 56 23 58 CD 34 E0 CD 46")
130 MEM$(&HE010,16)=HEXCHR$("E0 F3 1D 56 CD 34 E0 23 1D 20 F8 FB C9 FB EB 56")
140 MEM$(&HE020,16)=HEXCHR$("23 58 CD 34 E0 CD 46 E0 F3 1D CD 3D E0 72 23 1D")
150 MEM$(&HE030,16)=HEXCHR$("20 F8 FB C9 CD 46 E0 01 00 19 ED 51 C9 CD 50 E0")
160 MEM$(&HE040,16)=HEXCHR$("01 00 19 ED 50 C9 01 01 1A ED 78 E6 40 20 FA C9")
170 MEM$(&HE050,10)=HEXCHR$("01 01 1A ED 78 E6 20 20 FA C9")
180 '
190 POKE&H54,&H60,&HE0
200 MEM$(&HE060,16)=HEXCHR$("F5 C5 D5 E5 1E 02 21 80 E0 CD 3D E0 72 23 1D 20")
210 MEM$(&HE070,8)=HEXCHR$("F8 E1 D1 C1 F1 FB ED 4D")
220 MEM$(&HE080,2)=CHR$(0,0):'CLEAR WORK AREA
230 '
240 INIT
250 D$=CHR$(&HE4,&H54):'SET KEY VECT.
260 DUMMY$=USR0(D$)
270 '
280 PRINT RIGHT$("0"+HEX$(PEEK(&HE080)),2),: 'PRINT FUNC. PART
290 CODE=PEEK(&HE081)
300 PRINT RIGHT$("0"+HEX$(CODE),2),: 'PRINT ASCII PART
310 PRINT#0 CHR$(CODE) : 'DISP CHARACTER
320 IF CODE=&H1B THEN D$=CHR$(&HE4,&H52):DUMMY$=USR0(D$):END:' IF "ESC" END
330 GOTO 280

```



第

5

章

CTC



CTCは律儀なのである



## ■ 第5章

# CTCは律儀なのである・・・・・・・・

この章では CTC をやるのである。CTC とは Counter Timer Circuit の略である。

まず最初に断っておくが、残念なことに Non turbo の X1 には CTC, および SIO, DMA が付いていない。しかし、オプションの CZ-8BM2 を付ければ、そのボードには CTC と SIO が載っているという次第である (CTC は CZ-8BM2 以外でも、立体視ボード、FM 音源ボードに載っている)。この CZ-8BM2 は、RS-232C が 1 チャンネルとマウスインターフェイスが付いた、なかなかのボードである。CTC と SIO はそのボードの中で主役を演じているわけであるが、はっきり言って、このボードは turbo の CTC, SIO 周りの機能とほとんど同じなのである。唯一の違いは、I/O アドレスである。第 0 章でも示したが、turbo では、

SIO = 1F90<sub>H</sub> ~ 1F93<sub>H</sub>

CTC = 1FA0<sub>H</sub> ~ 1FA3<sub>H</sub>

だったのが、CZ-8BM2 では、

SIO = 1F98<sub>H</sub> ~ 1F9B<sub>H</sub>

CTC = 1FA8<sub>H</sub> ~ 1FAB<sub>H</sub>

となっただけである。ただしこれは工場出荷時の設定であるから、ショートピンを差し替えば turbo とまったく同じアドレスにすることも可能である。念のために表 5-1, 5-2 に示しておく。

表 5-1 SIO アドレス

アドレス	内 容	
1F90 <sub>H</sub> (1F98 <sub>H</sub> )	チャンネルA データポート	IN/OUT
1F91 <sub>H</sub> (1F99 <sub>H</sub> )	チャンネルA 制御語	IN/OUT
1F92 <sub>H</sub> (1F9A <sub>H</sub> )	チャンネルB データポート	IN/OUT
1F93 <sub>H</sub> (1F9B <sub>H</sub> )	チャンネルB 制御語	IN/OUT

表 5-2 CTC アドレス

アドレス	内 容	
1FA0 <sub>H</sub> (1FA8 <sub>H</sub> )	チャンネル0	IN/OUT
1FA1 <sub>H</sub> (1FA9 <sub>H</sub> )	チャンネル1(SIO チャンネルA用クロック)	IN/OUT
1FA2 <sub>H</sub> (1FAA <sub>H</sub> )	チャンネル2(SIO チャンネルB用クロック)	IN/OUT
1FA3 <sub>H</sub> (1FAB <sub>H</sub> )	チャンネル3	IN/OUT



そこでどのような方針でやるかであるが、まずは CTC の割り込みを使って「タイムシェアリングもどき」をやってしまうのである。本当はそんなたいそーなことではなくて、音楽を鳴らすだけなのだが、基本は同じなので大風呂敷を広げて 景気をつけてしまうのである。

なお、この章では参考文献 4 があると便利である。

## CTCの概略である

Z80 CTC には 0 番から 3 番まで、四つのチャンネルがある。ここでいうチャンネルとは、言ってみればそれぞれが独立した時計/カウンタのようなものである(本当は完全に独立しているわけではないのだが)。ざっと説明すると、X1 では、これら四つのチャンネルは二つのグループに分かれる。すなわちチャンネル 0, 3 とチャンネル 1, 2 である。

X1 ではチャンネル 0, 3 は主にタイマとカウンタに使われている。早い話が Z80 に対して一定時間ごとに割り込みをかけてくれるのである。一定時間ごとに割り込みをかけてくれるとどのようなメリットがあるかという、たとえばこれからやるような、音楽を鳴らす場合である。つまり、Z80 が別のことに熱中をしていても、CTC が「あなた、そろそろ PSG さんに次のデータを渡す時間ですよ」などと教えてくれるのである。つまりは秘書のようなものなのだ。秘書がいなければ、自分でしょっちゅう時計を見ていなければいけないし、うっかりすれば決められた時間を過ぎてしまうかもしれないのである。チャンネル 0, 3 はそのように使われているのである。

それに対して、X1 ではチャンネル 1, 2 は SIO につながれている。具体的に何をしているかという、RS-232C は 300 bps とか、1200 bps とかの転送速度があるが、そのための基準クロックを作っているのだ。すなわち、X1 で SIO を使おうとするなら、まずは CTC が第一関門になっているのである。そこいらへんのことは、次の第 6 章でつまびらかになるであろう。

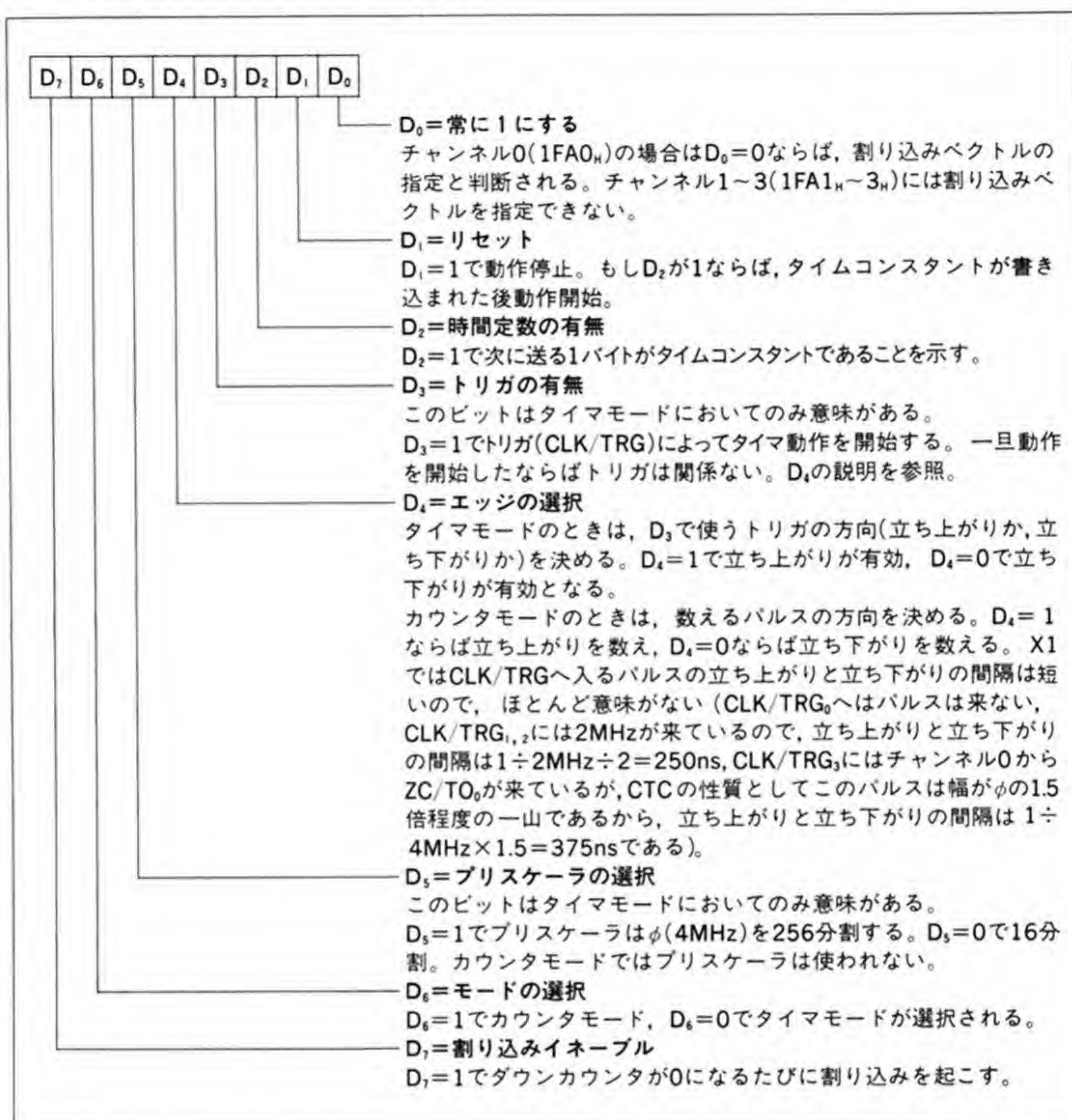
## CTCなものである

CTC のコマンドはそれほど複雑ではない。図 5-1 にあるようにコマンドは 1 バイトである。これを、四つのチャンネルに別々に送るわけである。指定した場合は、コマンドの直後に「タイムコンスタント」と呼ばれる 1 バイトを送ることもあるし、チャンネル 0 に対しては「割り込みベクトル」を送ることもあるが、結局はそれだけである。

まず、CTC のやることを、ごくごく簡単に説明しておく。最初に、図 5-2 を見ていただきたい。これは CTC の接続図である。「CLK/TRG<sub>n</sub>」とか「ZC/TO<sub>n</sub>」などがあるが、これは CTC の端子名である。「CLK/TRG<sub>n</sub>」とは「外部クロック/タイマ・トリガ」ということで、要するに各チャンネルごとにある入力である。「ZC/TO<sub>n</sub>」は「ゼロカウント/タイマ・アウト」で、こちらはそれぞれのチャンネルの出力に対応する(ただしチャンネル 3 用の ZC/TO<sub>3</sub> はピンの数の都合によって、省かれている)。



図 5-1 CTC のチャンネルコントロールワード

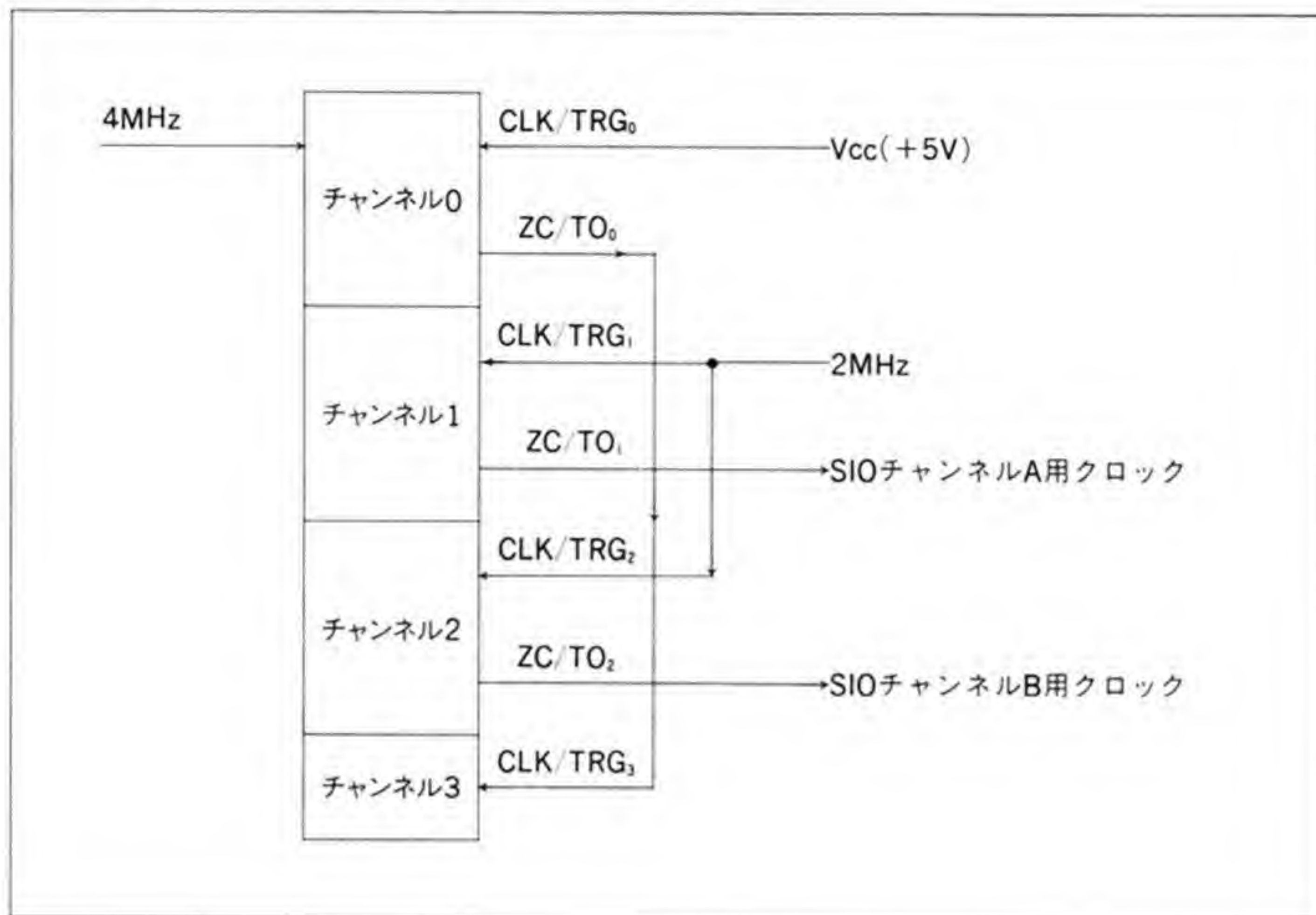


CTC のやることは、パルスを数えることである。数えられるパルスは図 5-2 の「CLK/TRG<sub>n</sub>」もしくは、システムクロックの  $\phi$  (X1 では 4MHz) である。CTC の各チャンネルには「ダウンカウンタ」というものがあり、最初は「ダウンカウンタ=タイムコンスタント」とするのである(タイムコンスタントは 0~255 を指定してやる)。各チャンネルは、それぞれがそれぞれのパルスを数えるのだが、何個かのパルスが来るたびに(1個、16個、256個の三つの場合がある)ダウンカウンタを一つずつ減らしていくのである。そして、ダウンカウンタが 0 になると(最初が 0 だったならば、256 回減らすことになる)、「ZC/TO<sub>n</sub>」から一つパルスを出すのである。つまり、パルスの数を  $1/n$  にするのである。もちろんそれだけではどーしょーもないので、「ZC/TO<sub>n</sub>」からパルスを出すのといっしょに、CPU に対して割り込みをかけたりもするのである。ま、これが CTC のやっていることである。

コマンドは各ビットごとに意味を持っているので、 $D_7$ から順に説明していくのである。



図 5-2 X1/turbo における CTC の各チャンネルの構成



$D_7$ はダウンカウンタが0になったときに、割り込みを起こさせるかどうかのフラグである。

$D_6$ はモードの設定である。それぞれのチャンネルに「カウンタモード」と「タイマモード」のどちらかを設定するわけであるが、「カウンタとタイマはどこがどー違うんでい」と、むっとする人が多いであろう。私もむっとしている。分かりやすく言うと、

「タイマモードでは、CTC に供給されているクロック (X1 では  $\phi=4\text{MHz}$ ) を数える」

「カウンタモードでは、CLK/TRG<sub>n</sub>という端子に来るパルス数を数える」

というだけのことである。

$D_5$ は $D_6$ でタイマモードを選択したときにのみ意味を持つ。これは、プリスケアラの指定で、例のダウンカウンタの値を勝手に16倍 ( $D_5=0$ )、もしくは256倍 ( $D_5=1$ )するものだと思えばよい。なぜそうなっているかというと、タイマモードのときの入力の $\phi$ は、一般的に4MHzなどという高い周波数なので、0~255までしかないダウンカウンタで数えても、数え終わるのにもっとも長くて $250\text{ns} \times 256 = 64\mu\text{s}$ しかなく、あまり使い道がないから、というZilogの親心だと考えられる（本当の親心ならダウンカウンタを2バイトで指定できるようにすりゃいいのに。ぶつぶつ）。まあ、ここはちょっとセコイ部分である。

$D_4$ はエッジの選択である。ここは (X1 では) 重要ではないので、図 5-1 を見ておくれ。

$D_3$ もX1ではあまり重要ではない。

$D_2$ は次にタイムコンスタントを送るかどうかのフラグである。第8章でやるZ80 DMA



の用語で言えば、ポインタビットというやつである。

$D_1$  はリセットであるが、ちょっと注意が必要である。つまり、CTC はリセットをかけられなかったなら、そのときやっている動作を終えてから、新しいコマンドに基づく動作を開始するのである。よって、リセットは大いに使うべきである。

$D_0$  は、コマンドの場合には、常に 1 にしておく。さもなくば、割り込みベクトルを指定されたものと解釈される。

## 割り込みベクトル

CTC では、割り込みベクトルは、チャンネル 0 に対してのみ指定できることになっている(つまり、割り込みベクトルを OUT できるのは  $1FA0_H$  番地にのみ)。それじゃ、他のチャンネルは割り込みをかけられないのかと、一瞬思ってしまうだろうがご安心。チャンネル 0 に対して指定すると、他の 3 チャンネルにも自動的に割り込みベクトルが割り当てられるのである。すなわち、連続した四つのベクトル (8 バイト) がまとめて指定されるのである。ただし、その 8 バイトには少々制限がある。それはどーゆーことかということ、チャンネル 0 のベクトルは、

$\&B????000$

のよーに、下 3 ビットが 0 になっていなければならないのだ。このとき各チャンネルのベクトルはそれぞれ、

1  $\rightarrow \&B????010$

2  $\rightarrow \&B????100$

3  $\rightarrow \&B????110$

となる。

だから単純に、「おっ、ここに 8 バイト分の空きがある。よしよし、CTC の割り込みテーブルに使ってしまおう」などということは許されないのだ。つまり、その 8 バイトの先頭が、

$\times \times \times 8_H$  か  $\times \times \times 0_H$  (下 3 ビットが 0)

でなければいけないのだ。この点、注意が必要である。

## CTCの使い方である

さて、あーだこーだと説明してきたが、X1 の場合はハード的な CTC の使い方が単純なので、悩むことはぜんぜんないのである。つまり、本来の CTC は  $D_3 = 1$  とすることで、「信号 (トリガ) が立ち上がってから、一定時間後に割り込みをかける」などという機能があるのだが、そんなことは X1 では使えないのである (使う必要もないだろう)。X1 の CTC の役割は極めて簡単明瞭で、次の三つに要約できる。

① 図 5-2 に示したように、X1 ではチャンネル 0 用の CLK/TRG<sub>0</sub> は  $V_{cc}$  (+5V) につながっている。すなわち、パルスがぜんぜん来ないので、チャンネル 0 はタイマモード



で使うしかない。その場合、チャンネル0は、 $4\mu\text{s}\sim 16.384\text{ms}$ ごとに割り込みをかけてくれる。この時間が短すぎるか、ちょうどピタリの間隔が得られないのならば、ZC/TO<sub>0</sub>がCLK/TRG<sub>3</sub>に接続されていることにより、チャンネル3をカウンタモードを使ってチャンネル0と組み合わせると、最長4.194304秒の間隔を得られる。また、チャンネル0, 3を別々に動かすことも可能である。

② チャンネル1, 2はそれぞれSIOのチャンネルA, Bのクロックにつながっているので、それなりに使ってやる。

ただし、気が向いたならばチャンネル0, 3と同じように、割り込みを起こさせることもできる (CLK/TRG<sub>1,2</sub>には2MHzが来ているのでチャンネル0よりも割り込み間隔を短くできる)。

③ OUT命令ではなく、IN命令でCTCにアクセスすると、そのときのダウンカウンタの値を得られる。ただし、これはあまり使い道があるとは考えられない。

そういうわけで実際に使ってみるのである。最初は、③に示したダウンカウンタを読み出すのをやってみる。これなら割り込みを使う必要がないので入門としてはおあつらえむきである。サンプルプログラムはリスト5-1である。

リスト5-1 ダウンカウンタを表示する

```
100 OUT &H1FA0,3:'チャンネル0 RESET
110 OUT &H1FA1,3:'チャンネル1 RESET
120 OUT &H1FA2,3:'チャンネル2 RESET
130 OUT &H1FA3,3:'チャンネル3 RESET
140 '
150 OUT &H1FA0, &B100111:OUT&H1FA0,200:'チャンネル0 SET
160 OUT &H1FA3,&B1000111:OUT&H1FA3,200:'チャンネル3 SET
170 '
180 PRINTINP(&H1FA0),INP(&H1FA3):GOTO 180
```

100～130行では縁起ものなので、各チャンネルに3をOUTしてリセットしている。次に150行でチャンネル0にOUTしているのは、&B00100111である。

順に見ていくと、

- 割り込みなし
- タイマモード
- プリスケアラは256分割
- 無視してよい
- 無視してよい
- 次にタイムコンスタントあり
- リセット ( $D_2 = 1$  だからタイムコンスタントを書き込まれた後で動作開始)
- $D_0$  だから1

となっている。だから次にOUTしている「200」はタイムコンスタントである。160行ではチャンネル3に&B01000111をOUTしている。これも順に見てゆくと、

- 割り込みなし
- カウンタモード



- プリスケアラは関係なし
- 無視してよい
- 関係なし
- 次にタイムコンスタントあり
- リセット ( $D_2 = 1$  だからタイムコンスタントを書き込まれた後で動作開始)
- $D_0$  だから 1

である。150 行と同じく、次に OUT している「200」はタイムコンスタントである。

このように CTC を設定してやると、どうなるかというと、

- 1) チャンネル 0 はタイマモードであるから、システムクロックの  $\phi$  (4MHz, 周期は 250 ns) を数える。プリスケアラが 256, タイムコンスタントが 200 であるから、ZC/TO<sub>0</sub> からパルスを出すのは、

$$250(\text{ns}) \times 256 \times 200 = 12.8(\text{ms})$$

となる。要するに、0.00000025秒周期のパルスをもとにして、0.0128秒周期のパルスを作っているのである。

- 2) チャンネル 3 はカウンタモードだから CLK/TRG<sub>3</sub> に来るパルス数を数える。そこには ZC/TO<sub>0</sub> がつながっているから、結局 0.0128 秒ごとに来るパルス数を数えることになる。タイムコンスタントは 200 だから、1 個パルスが来るたびに「200, 199, 198, …… , 2, 1」となる。1 の次はまた 200 で、延々とそれを繰り返すのである。ではぼちぼちと本筋へ入ってゆく。CTC の本筋といえば、当然ながらタイマ割り込みを使った「マルチタスク」である。理解に便利のように、CTC の設定を BASIC で書いてある。リスト 5-2 は、「割り込み実行ルーチン」のアセンブルリスト、リスト 5-3 がそれを使った例である。

リスト 5-3 を RUN すると「OK」と表示されたのち、画面に筋が走り出すであろう。しかしその他はまったく正常な BASIC のコマンド待ちの状態になっているはずである。試しにリストを取ってみることもできる。その場合はグラフィックが表示されなくなるが、**CTRL** + **D** を押せばまた表示されるはずである。これは遊びのようなサンプルであるが、タイマ割り込みを理解するにはちょうどよいだろう。

では実際に何が起きているのかを、ねっとりと解説してみる。

まず、220 行でチャンネル 0 に対してやっていることは、タイムコンスタントが 125 になったこと以外はリスト 5-1 の 150 行と同じである。次に 230 行で 58<sub>H</sub> を OUT しているが、これは  $D_0 = 0$  であるから割り込みベクトルの設定である。これにより、割り込みベクトルが、

チャンネル 0 は 58<sub>H</sub>

チャンネル 1 は 5A<sub>H</sub>

チャンネル 2 は 5C<sub>H</sub>

チャンネル 3 は 5E<sub>H</sub>

となる。240 行にあるチャンネル 3 の設定は  $D_7 = 1$  として割り込みを起こさせている点以外はこれもリスト 5-1 の 160 行と同じである。



## リスト 5-2 割り込み実行ルーチン

			.Z80	
			.PHASE	0E000H
E000	F5	;		
E001	C5	RUNG:	PUSH	AF
			PUSH	BC
E002	ED 4B E01D	;		
E006	78		LD	BC, (GADD)
E007	FE 40		LD	A, B
E009	30 03		CP	40H
			JR	NC, OKOUT ;BC >= 4000H?
E00B	01 4000	;		
			LD	BC, 4000H
E00E	3A E01F	OKOUT:	LD	A, (GPAT)
E011	ED 79		OUT	(C), A
E013	03		INC	BC
E014	ED 43 E01D		LD	(GADD), BC
E018	C1		POP	BC
E019	F1		POP	AF
E01A	FB		EI	
E01B	ED 4D		RETI	
E01D	4000	;		
E01F	FF	GADD:	DW	4000H
		GPAT:	DB	0FFH
		;		
			END	

## リスト 5-3 タイマ割り込みを使って BASIC と共走(turbo BASIC では動かない)

```

100 OUT &H1FA0,3:'チャンネル0 RESET
110 OUT &H1FA1,3:'チャンネル1 RESET
120 OUT &H1FA2,3:'チャンネル2 RESET
130 OUT &H1FA3,3:'チャンネル3 RESET
140 '
150 CLEAR &HE000
160 MEM$(&HE000,16)=HEXCHR$("F5 C5 ED 4B 1D E0 78 FE 40 30 03 01 00 40 3A 1F")
170 MEM$(&HE010,16)=HEXCHR$("E0 ED 79 03 ED 43 1D E0 C1 F1 FB ED 4D 00 40 FF")
180 MEM$(&H5E,2) =HEXCHR$("00 E0"):'ワリコミ TABLE SET
190 '
200 CLS 4:INIT
210 '
220 OUT &H1FA0,&B100111 :OUT &H1FA0,125:'チャンネル0 SET
230 OUT &H1FA0,&H58 :OUT &H1FA0,125:'ワリコミヘクトル SET
240 OUT &H1FA3,&B11000111:OUT &H1FA3,125:'チャンネル3 SET

```

さて、ちょっと前に戻って 180 行を見ていただきたい。ここで 005E<sub>H</sub> 番地からの 2 バイトの 00<sub>H</sub>, E0<sub>H</sub> を書き込んでいる。これがチャンネル 3 の割り込みテーブルである。仕掛けはこれだけである。これでチャンネル 3 により、

$$250(\text{ns}) \times 256 \times 125 \times 125 = 1(\text{秒})$$

ごとに割り込みが起き、そのたびに E000<sub>H</sub> 番地から始まるリスト 5-2 の割り込み実行ルーチンに飛んでくることになる。この機械語ルーチンを見てのとおり、1 バイト(8 ドット) ずつグラフィックにデータを書き込んでいくだけである。220 行もしくは 240 行の「125」を「1」に書き換えると書き込みが速くなるので、CLS4 と追いかけてこするのもまた楽しからずやである。

## 実技編である

タイマの割り込みを実用的に使う目的としては次の三つが挙げられるだろう。

- ① 一定の間隔(リズム)で何かをしたいとき



たとえば turbo BASIC の TEMPO 文ではタイマ割り込みの間隔を変えることにより音楽の演奏速度を変えている。また、ゲームの最中に流れている音楽が、敵キャラがたくさん出ているときは遅く、逆に少ないときは速くなったりするのをよく見かけるが、タイマを使えばそのようなことを比較的簡単に避けることができる。余談になるが、機械語で作られたプログラムでは、垂直帰線期間信号や垂直同期信号(I/O ポートの 1A01<sub>H</sub>)を使って一定のリズムで BGM を鳴らす手法がある。具体的にどうするかというと、これらの信号は 1/60 秒ぐらいの周期で High/Low を繰り返すので、

**頻繁にその信号を監視しておき、立ち上がりや立ち下がりを見つけることができれば、**

1/60 秒のリズムを捕えるのである。実際にやっているソフトは、アルシスソフトの「ウィバーン」、「リバイバー」などである。ううむ、なんという知恵者なことよ。

## ② 二つのことを同時にしたいとき

一番よい例が同じく turbo BASIC の MUSIC@文である。turbo BASIC では MUSIC@文中の演奏データは(どっかの)バッファに入れておくだけで、実際の演奏はその後に続く BASIC のコマンドを処理しながら、時々タイマ割り込みを使って、適当なデータを PSG に送り込むことによって、行なっている。もっと簡単に言ってしまうと、ゲームで PCG やグラフィックを動かしながら(リズムを狂わせずに)BGM や効果音を鳴らすことができる。

## ③ 遅い周辺機器を待たずに CPU をフルに使いたいとき

これは②の場合とかなり近いものだが、ひとまず区別しておく。これで一番よくあるのが(本体側ソフトウェアによる)プリンタバッファである。つまりプリンタの印字速度が遅いために、CPU がしこたま待たされるのを避けるためのものである。念のために言うが、プリンタから本体へは READY 信号というものが来ていて、結局は「本体さん、私ことプリンタにデータを 1 バイト送ってもいいよ」ということを示す信号なのである。この信号が「まだダメ」であるなら、CPU は待っていなければならないのだ。そこでプリンタバッファの動作を簡単に説明すると、まずはプリンタに送りたいデータはバッファに入れておく。そしてタイマ割り込みがかかるたびに READY 信号をチェックして、「送ってもいいよ」と言えば 1 バイト送ってやるのである。要するに「他の仕事をしながら、時々様子を見る」という使い方である。

## ④ そのものズバリに時計を作る

一定時間ごとに割り込みを起こしてくれるのだから、その回数を数えていれば時間が正確に分かるのである。短い方では、1/100 秒の測定などは朝飯前である。またメモリが許す限り長いタイマも作成可能である。たとえば何年、何世紀を 1/100 秒単位で計ることも簡単である。

そこでまず、②の BGM からやってみる。対比のためにリスト 5-4 にタイマを使わない場合も載せておく。

説明しよう。最初に並んでいる MEM\$文は機械語プログラムなどではなく、演奏データである。全部 BASIC なのだから配列にすりゃーいいのに、と思うだろうが、私はしたいよ



リスト 5-4 BASIC 版 BGM 実行ルーチン

```

100 CLEAR &HD000
110 MEM$(&HD000,12)=HEXCHR$("00 DD 01 01 08 0F 10 05 08 00 10 05")
120 MEM$(&HD00C,12)=HEXCHR$("00 A9 01 01 08 0F 10 05 08 00 10 05")
130 MEM$(&HD018,12)=HEXCHR$("00 7B 01 01 08 0F 10 05 08 00 10 05")
140 MEM$(&HD024,12)=HEXCHR$("00 DD 01 01 08 0F 10 05 08 00 10 05")
150 MEM$(&HD030,48)=MEM$(&HD000,48):'クリカエシ
160 MEM$(&HD060,12)=HEXCHR$("00 7B 01 01 08 0F 10 05 08 00 10 05")
170 MEM$(&HD06C,12)=HEXCHR$("00 65 01 01 08 0F 10 05 08 00 10 05")
180 MEM$(&HD078,12)=HEXCHR$("00 3E 01 01 08 0F 10 05 08 00 10 05")
190 MEM$(&HD084,12)=HEXCHR$("00 7B 01 01 08 0F 10 05 08 00 10 05")
200 MEM$(&HD090,48)=MEM$(&HD060,48):'クリカエシ
210 DEFINT A-Z:CLICK OFF:WIDTH 40:INIT
220 X=20:Y=12:LOCATE X,Y:PRINT "A";
230 MPS=&HD000:MPE=&HD0C0:MP=MPS:W=0
240 SOUND 7,&B111000:SOUND 8,0
250 '
260 GOSUB "SOUND"
270 S=STICK(0):T=STRIG(0)
280 IF S=0 THEN FOR D=0 TO 100:NEXT:GOTO 360
290 LOCATE X,Y:PRINTSPACES(1);:'ERASE CHAR
300 Y1=Y-((S+2)*3)+2
310 X1=X+((S+2) MOD 3)-1
320 IF (X1>=0) AND (X1<39) THEN X=X1
330 IF (Y1>=0) AND (Y1<23) THEN Y=Y1
340 LOCATE X,Y:PRINT "A";
350 '
360 IF T THEN END
370 GOTO 260
380 '
390 LABEL "SOUND"
400 IF W THEN W=W-1:RETURN:'WAIT
410 IF MP=MPE THEN MP=MPS
420 M1=PEEK(MP):MP=MP+1:M2=PEEK(MP):MP=MP+1
430 IF M1<&H10 THEN SOUND M1,M2:GOTO 410
440 IF M1=&H10 THEN W=M2:GOTO 400
450 IF M1=&H11 THEN W=M2*256:GOTO 400
460 PRINT "DATA ERROR":STOP

```

一にするのだ。で、演奏データの形式だが、これは2バイトが一組になっている。規則は簡単で、

- 1) 1バイト目が00<sub>H</sub>~0F<sub>H</sub>ならば、それはPSGのレジスタ番号である。PSGのその番号のレジスタに次の1バイトが書き込まれる。
- 2) 1バイト目が10<sub>H</sub>もしくは11<sub>H</sub>ならば、それはウェイト(時間つぶし)コマンドである(つまりPAUSE文のよーなもの)。10<sub>H</sub>の場合は次の1バイトに相当する時間、PSGには何も書き込まれない(消されもしない)。11<sub>H</sub>の場合は長いウェイトである。次の1バイト×256に相当する時間、PSGには何も書き込まれない。

これだけじゃなんだから、110行だけは解説しておく。まずは00<sub>H</sub>, DD<sub>H</sub>だから、「0番レジスタにDD<sub>H</sub>を書き込め」となる。次にある2組も同じよーなものである。結局この3組によって、チャンネルAから「ド」の音が音量15(08<sub>H</sub>, DF<sub>H</sub>)で出ることになる。しかしこれだけでは音にならない。つまり、一定時間以上鳴らし続けてやらなければ、人間の耳には「ド」に聞こえないのだ(「ブチッ」という音になってしまう)。そこで、次に10<sub>H</sub>, 05<sub>H</sub>でウェイトをおいてやる。その後一度チャンネルAの音量を0にして、もう一度ウェイトがある。これはMUSIC文を使ってメロディーを鳴らしたことがある人なら分かるように、音符と音符の間を区切るためのもの、つまりスタッカートである。リスト5-4のデー



タの仕組みは大体こんなふうになっている。

さて、このプログラムは実にセコくて、極めて原始的なことしかしていない。つまり、RUN すると画面の中央に「A」が表示され、「ドレミドードレミドーミファソミーミファソミー」というセコイ BGM がエンエンと繰り返される。しかし、お立ち会い。テンキーによって、「A」は上下左右に動くが、BGM はほとんど乱れずに続くのである。

さっさとネタをばらしてしまうが、これは「キャラクタを動かす」というプログラムがループになっているということを利用しているのだ。どーゆーことかという、まず、「ド」を出すならば、PSG をそのよーに設定してやる。つまり PSG のレジスタの 0, 1, 8 にそれぞれの値を書き込んでやる。“SOUND”に GOSUB すると、ウェイト中かもしくはウェイトコマンドが見つかるまでは RETURN しないので、最初に“SOUND”に GOSUB した段階で、すでに PSG は「ド」の音を出しているのである。

ウソだと思ったら（思うかな？）265 行に「END」を入れてみるといい。「ド」が鳴り続くはずである（止めるときは `CTRL+D`）。“SOUND”から RETURN した後は、キャラクタを動かす作業だけをすればよい。ただし、ループの時間（次に“SOUND”に GOSUB するまでの時間）を、あまり変えないようにすること。これは 280 行を見てもらえば分かると思うが、テンキーからの入力がなくキャラクタを動かす必要がない場合は、FOR 文で時間をつぶしている。その他の 290～340 行はキャラクタを動かしているだけのルーチンである。290 行は「A」を消している部分、300, 310 行はよくあるテクニックを使った新しい位置の計算である。320, 330 行は画面からはみ出さないようにチェックしてから、新しい位置を X, Y に代入している。340 行はその位置への表示である。

では、2 回目に“SOUND”に GOSUB したときはどうなっているかを見てみる。まず、前回のときはウェイトコマンドを見つけて、その後 400 行で  $W=W-1$  とした後 RETURN したのである（分かるかな？）。それは 440 行である。変数 W に 2 バイト目が代入された後に、さらに -1 されたのだから、2 回目に GOSUB してきた時点では W の値は 4 である。すると、400 行の「IF W～」に引っかかる。すなわち W から 1 引いて、W の値を 3 にしてから RETURN である。これと同じことが W が 0 になるまで繰り返されるわけだから、結局、

- 1 回目→PSG は「ド」を出し始める（W=4 で RETURN）
- 2 回目→何もしない（W=3 で RETURN）
- 3 回目→何もしない（W=2 で RETURN）
- 4 回目→何もしない（W=1 で RETURN）
- 5 回目→何もしない（W=0 で RETURN）
- 6 回目→W=0 だから、次のコマンドを捜す

ということになる。1 回目と 2 回目、2 回目と 3 回目、……の間には、270～370 行を実行している時間がそれぞれ挟み込まれているから、この場合、数えてみるとループ 5 回分の間「ド」が出ることになる。よく考えると分かることだが、440 行、450 行で「GOTO 400」とせずに「RETURN」とすると、ループ 6 回分になってしまうので、この手のプログラムを作るときは注意が必要である。



残りの部分を種明しすると、MP は演奏データ（アドレス）を指している変数である。MPS（データの始まり）、MPE（データの終わりの次）は、BGM を繰り返すためのもので、MP が MPE と等しくなったら、MP に MPS を代入してやって繰り返しに入るのである。後は 240 行であるが、これは本来ならば D000<sub>H</sub>からの音楽データの先頭に、

07<sub>H</sub>, 28<sub>H</sub>, 08<sub>H</sub>, 00<sub>H</sub>

というように入れてやるべきだったのだが、アドレスがズレてしまって美しくなくなるので手抜きをしたのだ。ま、以上である。

すでに気付いたことと思うが、このテクニックを使った場合、BASIC だろうが機械語だろうが、キャラクタを動かすルーチンの部分が多少なりとも複雑になってしまうと、BGM のリズム（テンポ）を一定に保つのは至難の技なのだ。まず第一に、ある局面になったら新しく敵キャラが出てくる場合や、さらにその敵キャラが増減する場合である。その度に、うまくヒマつぶしの時間を増減させてやらなければならないのだ。実にうっとうしい話ではないか。

**そこで CTC 様がリングに上がるのである。** タイマ割り込みによって変わるのは次の 2 点である。

- 1) GOSUB "SOUND" をループの中に置かなくてよい（決められた時刻になると、自動的に GOSUB してしまう）。
- 2) テンポを一定に保つためのヒマつぶしのルーチンを考えなくてもよい。

なんだなんだ！ 結局は「音無しのゲームを普通に作る」だけで済んでしまうのではないかっ！ ということでリスト 5-5, 5-6 である。リスト 5-5 はリスト 5-4 のプログラムをそのまま機械語に置き換えたようなものであるから説明はしない。

リスト 5-5 BGM 用割り込み実行ルーチン

			.Z80	
			.PHASE	0E000H
		;		
E000	F5	PLAY:	PUSH	AF
E001	C5		PUSH	BC
E002	D5		PUSH	DE
E003	E5		PUSH	HL
		;		
E004	2A E051	PLAY0:	LD	HL, (WC)
E007	7C		LD	A, H
E008	B5		OR	L
E009	28 06		JR	Z, NOWAIT
E00B	2B		DEC	HL
E00C	22 E051		LD	(WC), HL
E00F	18 39		JR	PEND
		;		
E011	2A E057	NOWAIT:	LD	HL, (MPE)
E014	ED 5B E053		LD	DE, (MP)
E018	B7		OR	A
E019	ED 52		SBC	HL, DE ;MPE>MP?
E01B	38 02		JR	C, OVER
E01D	20 04		JR	NZ, FETCH
E01F	ED 5B E055	OVER:	LD	DE, (MPS)
		;		
E023	EB	FETCH:	EX	DE, HL ;HL=MP
E024	7E		LD	A, (HL)
E025	23		INC	HL
E026	5E		LD	E, (HL)
E027	23		INC	HL
E028	22 E053		LD	(MP), HL ;STORE MP
		;		
E02B	FE 10		CP	16



E02D	30 0A		JR	NC,SETWT ;SET WAIT COUNTER
		;		
E02F	01 1C00		LD	BC,1C00H
E032	ED 79		OUT	(C),A ;SET REG. NUMBER
E034	05		DEC	B
E035	ED 59		OUT	(C),E
E037	18 D8		JR	NOWAIT
		;		
E039	FE 10	SETWT:	CP	16
E03B	20 04		JR	NZ,SETWT1 ;BIG WAIT
E03D	16 00		LD	D,0
E03F	18 03		JR	SETWT2
E041	53	SETWT1:	LD	D,E
E042	1E 00		LD	E,0
E044	ED 53 E051	SETWT2:	LD	(WC),DE
E048	18 BA		JR	PLAY0
		;		
E04A	E1	PEND:	POP	HL
E04B	D1		POP	DE
E04C	C1		POP	BC
E04D	F1		POP	AF
E04E	FB		EI	
E04F	ED 4D		RETI	
		;		
E051	0000	WC:	DW	0000H ;WAIT COUNTER
E053	0000	MP:	DW	0000H ;MUSIC DATA POINTER
E055	0000	MPS:	DW	0000H ;MUSIC START
E057	0000	MPE:	DW	0000H ;MUSIC END
		;		
			END	

リスト 5-6 機械語版 BGM 実行ルーチン(turbo BASIC では動かない)

100	'100 - 130 ニハ, リスト5-1 ノ 100 - 130 キ* ヲウ ヲ ソノママ モツテ クル。
140	'
150	CLEAR &HE000
160	MEM\$(&HE000,16)=HEXCHR\$("F5 C5 D5 E5 2A 51 E0 7C B5 28 06 2B 22 51 E0 18")
170	MEM\$(&HE010,16)=HEXCHR\$("39 2A 57 E0 ED 5B 53 E0 B7 ED 52 38 02 20 04 ED")
180	MEM\$(&HE020,16)=HEXCHR\$("5B 55 E0 EB 7E 23 5E 23 22 53 E0 FE 10 30 0A 01")
190	MEM\$(&HE030,16)=HEXCHR\$("00 1C ED 79 05 ED 59 18 D8 FE 10 20 04 16 00 18")
200	MEM\$(&HE040,16)=HEXCHR\$("03 53 1E 00 ED 53 51 E0 18 BA E1 D1 C1 F1 FB ED")
210	MEM\$(&HE050,16)=HEXCHR\$("4D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00")
220	'
230	'230 - 330 ニハ, リスト5-4 ノ 100 - 200 キ* ヲウ ヲ RENUMBER シテ モツテ クル。
340	'
350	MEM\$(&HE051,8)=HEXCHR\$("00 00 00 D0 00 D0 C0 D0")
360	MEM\$(&H5E,2) =HEXCHR\$("00 E0")
370	'
380	SOUND 7,&B111000:SOUND 8,0
390	OUT &H1FA0,&B111 :OUT &H1FA0,100:'チャンネル0 SET
400	OUT &H1FA0,&H58 :'ワリコミヘ*クトル SET
410	OUT &H1FA3,&B11000111:OUT &H1FA3,25 :'チャンネル3 SET

## 結論である

以上なわけである。この章のサンプルは、CTC の I/O アドレスを 1FA8<sub>H</sub>~1FAB<sub>H</sub>にずらしてやれば、CZ-8BM2 を装着した X1 でも動くはずである。ぜひとも試していただきたい。また、リスト 5-6 では PSG のチャンネル B, C を使っていないので、第 10 章のリスト 10-3 を少々変えて、チャンネル A をいじらないようにすると、BGM と銃声が共存共栄するのである。具体的にどうするかというと、リスト 10-3 の 120 行にある「&B000111」を「&H001110」にすればよいだけなのだ。これでチャンネル A はトーン、B と C はノイズになる。どーだ、すごいだろう。止めるときは、「OUT &H1FA3, 3」である。

あと CTC の使い方としては時計が代表的なものであるが、以上のことを理解できればあっという間に作れてしまうだろう。



なお、あまり勧められた手ではないが、turbo BASIC ではちょちょいのちょいで簡単に CTC の割り込みを使う手がある。それは、F81E<sub>H</sub>にある割り込みベクトルを書き換えるという手である。割り込み間隔は TEMPO 文で変更できる。この場合もう一工夫しないと PLAY 文が使えなくなってしまう、などの欠点があるが、手軽さには捨てがたいものがある。実例は 12 章（カラーイメージボード）に載せてある。

CTC の使い方は限られているが、その応用例は広いのである。そして、CTC だけではどうにもならないが、大事な基本技として必修なのである。





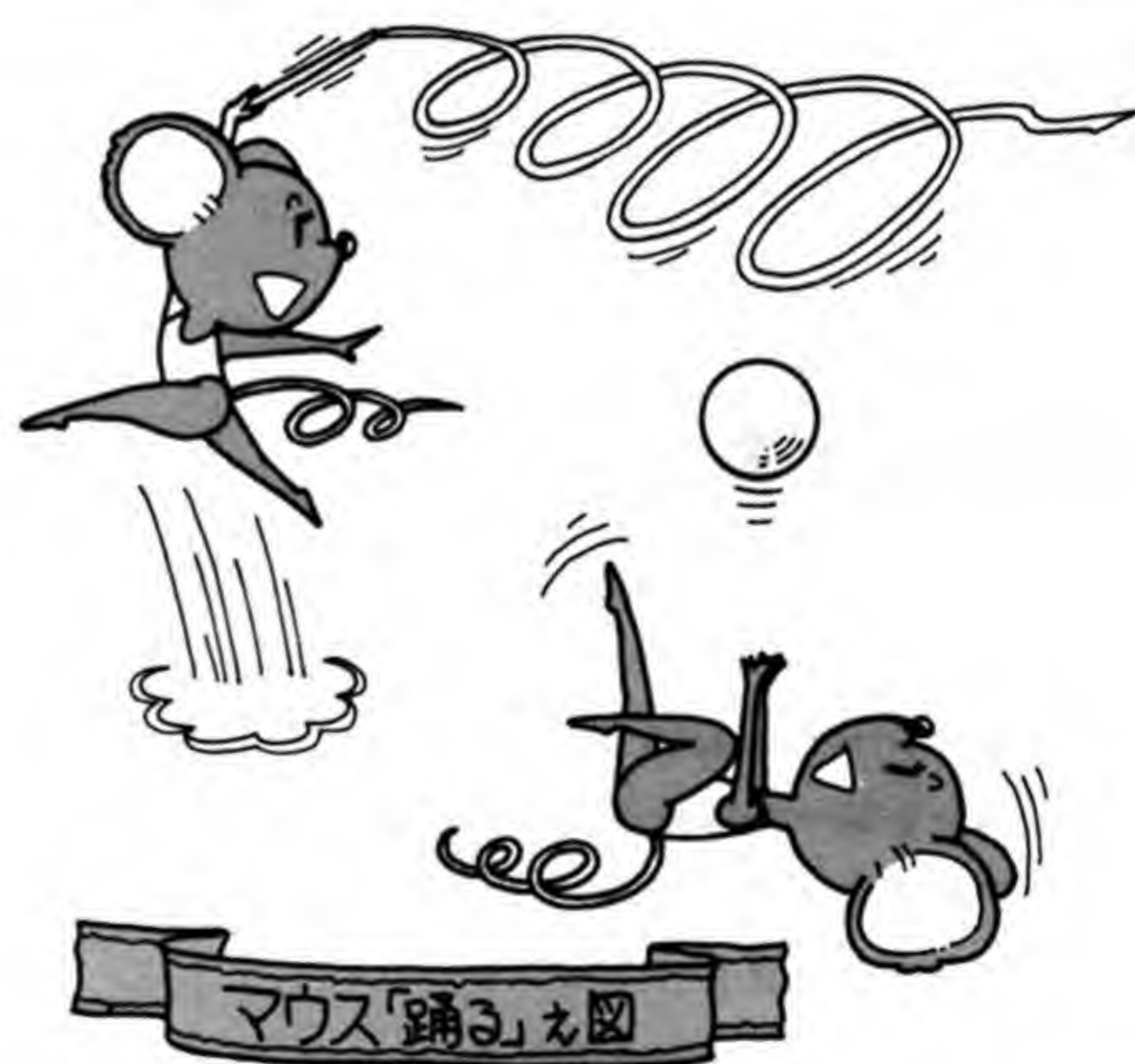


第

6

章

SIO



SIOでマウスである



## ■ 第6章

# SIOでマウスである・・・・・・・・

この章で取り上げる Z80 SIO (Serial Input/Output) は、シリアル通信を主な業務としている石である。そして、この石はやたらと複雑で強力なのである。これはとりもなおさず、シリアル通信というものが複雑怪奇であることに由来する。実際にパソコンで通信をする場合には、BASIC の OPEN 文の通信パラメータで指定するだけの機能があれば済むのだが、SIO にはそれ以上の機能が盛り込まれているのである。そういうわけで、不本意ながら SIO の使い方を細かく説明するわけにはいかない。もしも SIO の機能をみっちり知りたいならば、参考文献 4 を読んでいただきたい。

さて、実は Z80 SIO といっても三つのバージョン (ボンディングオプション) があるのだ。すなわち、

SIO/0, SIO/1, SIO/2

である。X1 に使われているのはそのうちの SIO/0 である。これら三つのものの違いは、チャンネル B に対する制限で、早い話がピン数 (足の数) に限りがあるのでそうになっているのである (X1 ではチャンネル B はマウスに接続するわけだから、まったく問題ない)。

SIO にはチャンネル A, B の二つのチャンネルがあり、X1 ではチャンネル A が RS-232C の通信用で、チャンネル B が今書いたようにマウス用になっている。そして、まずはこの章でチャンネル B=マウスを先にやった後で、次の第7章でチャンネル A の通信関係へと進むつもりである。

## マウスである

X1 のマウスは RS-232C のシリアルマウスで、データの方法はマウス→X1 の一方通行である。マウスから送ってくるデータは、

- ① ボーレートは 4800 bps
- ② コントロール信号を立ち下げる (H→L) とデータを送ってくる
- ③ データは一組が 10 ビット構成で、  
    スタートビット = 1 ビット  
    データビット = 8 ビット  
    ストップビット = 1 ビット

の 10 ビットが 3 組

となっている。③でいきなりスタートビットとかストップビットとかが出てきたが、これがシリアル通信のうっとーしいところである。具体的にどういうシロモノかというと、「はい、今からデータが行きますよー」、「はい、これでデータは終わりです」とかいう区切りなのだ。一般の通信では、データが 7 ビットだったり、ストップビットが 1.5 もしくは 2



図 6-1 マウスからのデータの内容

1 バイト目	ステータス	D <sub>0</sub>	スイッチ1の状態	1でON, 0でOFF
		D <sub>1</sub>	スイッチ2の状態	
		D <sub>2</sub>	——	
		D <sub>3</sub>	——	
		D <sub>4</sub>	Xのオーバーフロービット(128以上のとき)	
		D <sub>5</sub>	Xのアンダーフロービット(-129以下のとき)	
		D <sub>6</sub>	Yのオーバーフロービット(128以上のとき)	
		D <sub>7</sub>	Yのアンダーフロービット(-129以下のとき)	
2 バイト目	X方向の移動量	-128~+127	D <sub>7</sub> が符号ビットである 読み出した1バイトのデータがXならば, D <sub>7</sub> =0→移動量=X≥0 D <sub>7</sub> =1→移動量=X-256<0	
3 バイト目	Y方向の移動量	-128~+127		

ビット（長さが1.5倍もしくは2倍なのだ）のこともある。さて、結局マウスからは3バイトのデータを送ってくるわけである。その内訳は図 6-1 である。

まずステータスについて説明する。D<sub>0</sub>, D<sub>1</sub> のスイッチは自明だからよいだろう。2ビット空いて D<sub>4</sub>~D<sub>7</sub> がオーバー/アンダーフロービットである。これはマウス内のカウンタがあふれたということで、たとえば D<sub>4</sub> = 1 だったなら、マウスが「おいらはよく分かんないけど、X 方向に 128 以上動いちゃった。だから 2 番目のデータの『X 方向の移動量』はあてにならないよ」と言っているわけである。以下同様。

次に X/Y 方向の移動量であるが、一番大事なことは、前回データを送ったときからの相対距離だということである。turbo BASIC の MOUSE(7), MOUSE(8) に相当するものだから、BASIC のマニュアルを見ておくれ。

ではリスト 6-1 によって、具体的にマウスを使う方法を示す。恐ろしいことにオール BASIC である。心得のある人ならば「4800 bps で 30 ビットを送ってくるんだから、30÷4800=0.00625 秒じゃねーか。BASIC で間に合うの？」と思うであろう。しかし目出たいことに、Z80 SIO はバッファを三つ分（この場合だと 3 バイト分）持っているのだ。だからどんなにゆっくりやっても、マウスからのデータならば取りこぼしがないのである。楽ちん楽ちん。

リスト 6-1 マウスの使い方、もしくはバッファは 3 拍子

```

100 GOSUB "SETCTC&SIO"
110 '
120 OUT &H1F93,5:OUT &H1F93,0 : 'コントロール シンゴウヲ H ニ スル
130 PAUSE 0 : 'チヨット タケ マツ (キフン)
140 OUT &H1F93,5:OUT &H1F93,2 : 'REQUEST(コントロール シンゴウ=RTS ノ タチサケ)
150 FOR I=1 TO 3
160 OUT &H1F93,0:BBB=INP(&H1F93) AND 1 : 'BUFF ニ データ カ アルカ?
170 IF BBB=0 THEN 160
180 DDD=INP(&H1F92)
190 PRINT DDD
200 NEXT

```



```

210 PRINT STRING$(10,"=")
220 GOTO 120
230 '
240 LABEL"SETCTC&SIO"
250 OUT &H1FA2,&B1000111:'CTC チャンネル 3 SET
260 OUT &H1FA2,26          : 'TIME CONSTANT
270 '
280 RESTORE "SIODATA"
290 FORI=1TO15
300   READ D:OUT&H1F93,D
310 NEXT
320 RETURN
330 '
340 LABEL"SIODATA"
350 DATA &B00011000      : 'RESET
360 DATA 1,0              : 'WR0=00H
370 DATA 2,00H            : 'WR2=00H=INT. VECTOR (turbo ノ ハアイ)
380 DATA 4,&B01000100     : 'WR4=44H='クロック ハ *16' & 'ストップ ビット=1 ビット'
390 DATA 5,00H            : 'WR5=00H
400 DATA 6,00H            : 'WR6=00H
410 DATA 7,00H            : 'WR7=00H
420 DATA 3,&B11000001     : 'WR3=C1H='DATA=8ビット' & 'イネーブル レシーバ'

```

さて、リスト 6-1 はしょっぱなから 240 行へ GOSUB している。これは CTC と SIO の設定である。まずは 250～260 行で SIO のチャンネル B 用のクロックを作っている CTC のチャンネル 2 を、「カウンタモードでタイムコンスタントは 26」にする。これによってチャンネル 2 は 2MHz のクロックを数えるわけだから、

$$\frac{1}{2 \times 10^6 (\text{Hz})} \times 26 = 13 \times 10^{-6} (\text{秒})$$

ごとにパルスを出す。SIO は、このパルスを 16 もしくは 64 分周してボーレートにするのである。この場合は、16 分周すれば、

$$13 \times 10^{-6} \times 16 = 208 \times 10^{-6} (\text{秒})$$

となり、結局これは、

$$\frac{1}{208 \times 10^{-6}} = 4807.69 \cdots \approx 4800$$

というしだいである。きっちり 4800 である必要はないのである。次は 280 行からであるが、ここで SIO を設定している。やっていることは、

- キャラクタ長は 8 ビット
- クロックは CTC からのパルスを 16 分周
- ストップビットは 1
- 割り込みなし

などである。

ここを RETURN した後に、いよいよメインルーチンである。120～140 行ではコントロール信号を立ち下げているのだが、マウスへのコントロール信号はチャンネル B の RTS なのである。RTS とは「Request To Send」で、要するに「送ってきてちょうだい」である。これが H → L になると、マウスは律儀に 3 バイトのデータを送ってくる。130 行に PAUSE 文があるが、これは気分の問題である。BASIC ならばどうでもよいのだが、機械語でやる場合はそうはいかない。ある程度の時間「H」にしておいてから「L」にしなければ



ばならないのだ。実にやっかいなものだ。

150~200 行の FOR~NEXT 文で 3 バイトのデータを SIO のバッファから読み出して表示している。160, 170 行は「バッファにデータが入っているか」の判定ルーチン, 180 行がデータの読み出しである。プログラムを RUN すると 3 バイトのデータを表示していく。マウスを動かしたり, ボタンを押したりすると数字が変わるであろう。

## マウスの実技である

悲しいことであるが, 世の中には「マウスよりもトラックボールがいいのだ」, 「うんや, ライトペンこそが一等賞!」, 「てやんでい。タッチパネルを忘れてもらっちゃ困るぜい」などと言う人達がいる。あなた達に神の御許しがありますよーに, というわけで私はマウス党なのである。

そこでリスト 6-3, 6-4 をさっさと実行していただきたい。ちなみにリスト 6-3 のソースリストはリスト 6-2 となっている。リスト 6-3 はチェックサムと CRC 付きのダンプリストである。入力に際しては付録 A のダンプリストチェックプログラムを参照のこと。

リスト 6-2 マウスドライバ, もしくはブリマドンナ

			.Z80	
			.PHASE	0EA00H
1F90			;	
1FA0		ZSIO	EQU	1F90H
0018		ZCTC	EQU	1FA0H
		PSIZE	EQU	14+8+2 ;PARAMETER SIZE
		;		
EA00	EB	MOUSE:	EX	DE,HL
EA01	5E		LD	E,(HL)
EA02	23		INC	HL
EA03	56		LD	D,(HL)
EA04	EB		EX	DE,HL
		;		
EA05	22 EB29		LD	(RETV),HL
		;	SAVE RETURN VALUE AREA	
EA08	11 EB31		LD	DE,X1
EA0B	01 0018		LD	BC,PSIZE
EA0E	ED B0		LDIR	;COPY
		;		
EA10	3E 50		LD	A,50H
EA12	32 EC61		LD	(PATCH1+1),A
EA15	3E 29		LD	A,29H
EA17	32 EC7A		LD	(PATCH2),A
EA1A	3E 28		LD	A,28H
EA1C	32 EAAC		LD	(PATCH3),A
EA1F	21 ECA6		LD	HL,ARW8
EA22	22 EB2F		LD	(DFMC),HL
EA25	3A EB48		LD	A,(MODE)
EA28	0F	RRCA		
EA29	38 17	JR	C,W180	;CHECK BIT0
		;		
EA2B	47		LD	B,A
EA2C	3E 28		LD	A,28H
EA2E	32 EC61		LD	(PATCH1+1),A
EA31	3E 00		LD	A,000H
EA33	32 EC7A		LD	(PATCH2),A
EA36	3E 18		LD	A,18H
EA38	32 EAAC		LD	(PATCH3),A
EA3B	21 EDD4		LD	HL,ARW4
EA3E	22 EB2F		LD	(DFMC),HL
EA41	78		LD	A,B
		;		
EA42	0F	W180:	RRCA	
EA43	30 04		JR	NC,MSG0
EA45	CD EBD9		CALL	PSET
				;JUST GRAPHIC



EA48	C9		RET		
EA49	2A EB31		MSGO: LD	HL, (X1)	;COPY POSITION
EA4C	22 EB2B		LD	(X0), HL	
EA4F	2A EB33		LD	HL, (Y1)	
EA52	22 EB2D		LD	(Y0), HL	
EA55	CD EADC		CALL	CIN	;CHECK IN
EA58	E6 03		AND	3	
EA5A	28 05		JR	Z, BEGIN	
EA5C	3E 02		LD	A, 2	
EA5E	C3 EB16		JP	ERR0	
EA61	CD EBA5		BEGIN: CALL	SETMS	;MOUSE INIT
EA64	CD EBD9		CALL	PSET	;SHOW CURSOR
EA67	FD 21 EB39		LD	IY, MSDATA	
EA6B	CD EB49		CALL	MSIN	
EA6E	3A EB39		BEGIN0: LD	A, (MSDATA)	
EA71	F5		PUSH	AF	
EA72	FD 21 EB39		LD	IY, MSDATA	;IY=POINTER
EA76	CD EB49		CALL	MSIN	;READ MOUSE
EA79	F1		POP	AF	
EA7A	B7		OR	A	
EA7B	20 11		JR	NZ, BUT2	
EA7D	21 EB35		LD	HL, BMASK	;CONDITION
EA80	3A EB39		LD	A, (MSDATA)	;GET STAT
EA83	E6 03		AND	3	
EA85	06 04		LD	B, 4	
EA87	BE		BUT0: CP	(HL)	;CHECK BUTTON
EA88	23		INC	HL	
EA89	CA EB11		JP	Z, BUTTON	;BUTTON HIT
EA8C	10 F9		BUT1: DJNZ	BUT0	
EA8E	21 EB3A		BUT2: LD	HL, MSDATA+1	
EA91	7E		LD	A, (HL)	
EA92	23		INC	HL	
EA93	B6		OR	(HL)	
EA94	28 D8		JR	Z, BEGIN0	;X=Y=0 THEN NO-O
EA96	3A EB3A		LD	A, (MSDATA+1)	
EA99	CD EB0A		CALL	XSIGN	
EA9C	2A EB31		LD	HL, (X1)	
EA9F	19		ADD	HL, DE	
EAA0	22 EB2B		LD	(X0), HL	
EAA3	3A EB3B		LD	A, (MSDATA+2)	
EAA6	CD EB0A		CALL	XSIGN	
EAA9	3E 01		LD	A, 1	
EAAB	BB		CP	E	
EAAC	28 04		PATCH3: JR	Z, OK1	;1 -> 1 (-1 -> -1)
EAAE	CB 2A		;OR JR	OK1	(WIDTH 40)
EAB0	CB 1B		SRA	D	
EAB2	2A EB33		RR	E	;DE=DE/2 with sign
EAB5	19		OK1: LD	HL, (Y1)	
EAB6	22 EB2D		ADD	HL, DE	
			LD	(Y0), HL	
EAB9	CD EADC		;CHRCK IN?		
EABC	FE 03		CALL	CIN	
EABE	28 AE		CP	3	
EAC0	F5		JR	Z, BEGIN0	;X AND Y OUT
EAC1	CD EBD9		PUSH	AF	
EAC4	F1		CALL	PSET	;ERASE CURSOR
EAC5	0F		POP	AF	
EAC6	38 06		RRCA		
EAC8	2A EB2B		JR	C, YCHECK	;X=OB THEN CHECK
EACB	22 EB31		LD	HL, (X0)	;COPY POSITION
EACE	0F		LD	(X1), HL	
EACF	38 06		YCHECK: RRCA		
EAD1	2A EB2D		JR	C, SHOWC	
			LD	HL, (Y0)	



EAD4	22 EB33		LD	(Y1),HL	
EAD7	CD EBD9	;	SHOWC:	CALL	PSET
EADA	18 92		JR	BEGIN0	;SHOW NEW CURSOR
					;AGAIN
EADC	AF	;	CIN:	XOR	A
EADD	ED 5B EB44		LD	DE,(YU)	
EAE1	2A EB2D		LD	HL,(Y0)	
EAE4	B7		OR	A	
EAE5	ED 52		SBC	HL,DE	
EAE7	38 09		JR	C,XIN	
		;			
EAE9	ED 5B EB2D		LD	DE,(Y0)	
EAED	2A EB46		LD	HL,(YD)	
EAF0	ED 52		SBC	HL,DE	
		;			
EAF2	17	;	XIN:	RLA	
EAF3	ED 5B EB40		LD	DE,(XL)	
EAF7	2A EB2B		LD	HL,(X0)	
EAFA	B7		OR	A	
EAFB	ED 52		SBC	HL,DE	
EAFD	38 09		JR	C,XIN0	
		;			
EAFF	ED 5B EB2B		LD	DE,(X0)	
EB03	2A EB42		LD	HL,(XR)	
EB06	ED 52		SBC	HL,DE	
EB08	17		XIN0:	RLA	
EB09	C9		RET		
		;			
EB0A	5F	;	XSIGN:	LD	E,A
EB0B	16 00		LD	D,0	
EB0D	07		RLCA		
EB0E	D0		RET	NC	
EB0F	15		DEC	D	;D=FFH or 00H
EB10	C9		RET		
		;			
EB11	AF	;	BUTTON:	XOR	A
EB12	18 02		JR	ERR0	
		;			
EB14	3E 01	;	ERR:	LD	A,1
EB16	32 EB3C		ERR0:	LD	(ECODE),A
EB19	CD EBD9		CALL	PSET	;ERASE CURSOR
EB1C	ED 5B EB29		LD	DE,(RETV)	
EB20	21 EB31		LD	HL,X1	
EB23	01 0018		LD	BC,PSIZE	
EB26	ED B0		LDIR		
EB28	C9		RET		
		;			
EB29			RETV:	DS	2 ;RETURN VALUE ADD.
EB2B			X0:	DS	2 ;BUFF.
EB2D			Y0:	DS	2 ;BUFF.
EB2F			DFMC:	DS	2 ;DEFAULT CURSOR
		;			
EB31			X1:	DS	2 ;X POSITION
EB33			Y1:	DS	2 ;Y POSITION
EB35			BMASK:	DS	4 ;BUTTON CONDITION
EB39			MSDATA:	DS	3 ;MOUSE DATA
EB3C			ECODE:	DS	1 ;ERROR CODE
EB3D			MC:	DS	2 ;MOUSE CURSOR PAT ADD
EB3F			MCCOL:	DS	1 ;MOUSE CURSOR COLOR
EB40			XL:	DS	2 ;X UNDER LIMIT
EB42			XR:	DS	2 ;X UPPER LIMIT
EB44			YU:	DS	2 ;Y UNDER LIMIT
EB46			YD:	DS	2 ;Y UPPER LIMIT
EB48			MODE:	DS	1 ;WIDTH AND WRITE PAT
		;			
EB49	01 1F93	;	MSIN:	LD	BC,ZSIO+3
EB4C	3E 05		LD	A,5	; ->WR5
EB4E	ED 79		OUT	(C),A	;WR5 SELECT
EB50	AF		XOR	A	
EB51	ED 79		OUT	(C),A	;RTS High
		;			
EB53	16 80		LD	D,80H	
EB55	15		MSINW:	DEC	D ;WAIT
EB56	20 FD		JR	NZ,MSINW	
		;			
EB58	3E 05		LD	A,5	
EB5A	ED 79		OUT	(C),A	;WR5 SELECT
EB5C	3E 02		LD	A,2	
EB5E	ED 79		OUT	(C),A	;RTS Low



```

EB60      1E 05
EB62      FD 22 EB80
EB66      16 03

EB68      CD EB82
EB6B      38 09
EB6D      FD 77 00
EB70      FD 23
EB72      15
EB73      20 F3
EB75      C9

EB76      FD 2A EB80
EB7A      1D
EB7B      20 E9

EB7D      C3 EB14

EB80

EB82      01 1F93
EB85      21 0000
EB88      2B
EB89      7C
EB8A      B5
EB8B      28 0D

EB8D      AF
EB8E      ED 79
EB90      ED 78
EB92      1F
EB93      30 F3

EB95      0B
EB96      ED 78
EB98      B7
EB99      C9

EB9A      D5
EB9B      CD EBA5
EB9E      D1

EB9F      2B
EBA0      7C
EBA1      B5
EBA2      20 FB

EBA4      C9

EBA5      F3
EBA6      CD EBAE
EBA9      CD EBBA
EBAC      FB
EBAD      C9

EBAE      01 1FA2
EBB1      3E 47
EBB3      ED 79
EBB5      3E 1A
EBB7      ED 79
EBB9      C9

EBBA      01 1F93
EBBD      21 EBCA
EBC0      16 0F
EBC2      7E
EBC3      23
EBC4      ED 79
EBC6      15
EBC7      20 F9

EBC9      C9

EBCA      18
EBCB      01 00
EBCD      02 70
EBCF      04 44

```

```

;
LD      E,5      ;RETRY COUNT
LD      (IYBUFF),IY
MSRTRY: LD      D,3      ;COUNT
;
MSINL:  CALL     MSGET
JR      C,FAIL
LD      (IY+0),A
INC     IY
DEC     D
JR      NZ,MSINL
RET

;
FAIL:   LD      IY,(IYBUFF)
DEC     E
JR      NZ,MSRTRY
;
JP      ERR
;
IYBUFF: DS      2
;
MSGET:  LD      BC,ZSIO+3
LD      HL,0000H
MSGET0: DEC     HL
LD      A,H
OR      L
JR      Z,LATE
;
XOR     A
OUT     (C),A      ;RR0 SELECT
IN      A,(C)
RRA                      ;Bit0 ON?
JR      NC,MSGET0
;
DEC     BC      ;BC=1F92H
IN      A,(C)      ;GET DATA
OR      A      ;RESET CARRY FLAG
RET

;
LATE:   PUSH    DE
CALL    SETMS
POP     DE
;
;HL=0000H
LATE0:  DEC     HL
LD      A,H
OR      L
JR      NZ,LATE0
;
RET

;
;::::::::::::::::::::::::::
SETMS:  DI
CALL    SETCTC
CALL    SETSIO
EI                      ;INITIALIZE
RET

;
SETCTC: LD      BC,ZCTC+2      ;CTC2=CHANNEL B
LD      A,01000111B      ;MODE
OUT     (C),A
LD      A,26      ;TIME CONSTANT
OUT     (C),A
RET

;
SETSIO: LD      BC,ZSIO+3      ;CHANNEL B
LD      HL,SIODAT
LD      D,15      ;COUNT
SIOL:   LD      A,(HL)
INC     HL
OUT     (C),A
DEC     D
JR      NZ,SIOL
;
RET

;
SIODAT: DB      00011000B
DB      1,00H
DB      2,70H
DB      4,01000100B

```



EBD1	05 00	DB	5,00H	
EBD3	06 00	DB	6,00H	
EBD5	07 00	DB	7,00H	
EBD7	03 C1	DB	3,11000001B	
;				
;HL=Y1,BC=X1,DE=addr				
EBD9	ED 4B EB31	PSET:	LD BC,(X1)	
EBDD	2A EB33		LD HL,(Y1)	
EBE0	ED 5B EB3D		LD DE,(MC)	
EBE4	7A		LD A,D	
EBE5	B3		OR E	;DE=0000 ?
EBE6	20 04		JR NZ,PSET1	
EBE8	ED 5B EB2F		LD DE,(DFMC)	;DEFAULT CURSOR
EBEC	3A EB3F	PSET1:	LD A,(MCCOL)	
EBEF	CD EBF8		CALL PAT	
EBF2	C9		RET	
;				
EBF3		PATD:	DS 2	
EBF5		PATB:	DS 2	
EBF7		COLOR:	DS 1	
;				
;HL=Y1,BC=X1,DE=PATTERN ADDR.				
;				
EBF8	ED 53 EBF3	PAT:	LD (PATD),DE	;STORE PAT. ADDR.
EBFC	0F		RRCA	
EBFD	0F		RRCA	
EBFE	32 EBF7		LD (COLOR),A	;STORE COLOR
;				
EC01	CD EC6E		CALL XYADDR	
EC04	22 EBF5		LD (PATB),HL	;STORE BEGIN ADDR
;				
;discard MASK				
EC07	6A		LD L,D	
EC08	26 00		LD H,0	
EC0A	29		ADD HL,HL	
EC0B	ED 5B EBF3		LD DE,(PATD)	
EC0F	19		ADD HL,DE	
EC10	11 EBF3		LD DE,PATD	
EC13	01 0002		LD BC,2	
EC16	ED B0		LDIR	
;				
EC18	2A EBF3	PATGO:	LD HL,(PATD)	
EC1B	ED 4B EBF5		LD BC,(PATB)	
EC1F	78		LD A,B	
EC20	E6 C0		AND 0C0H	
EC22	5F		LD E,A	;E=40H,80H or C0H
EC23	57		LD D,A	
EC24	E2 EC29		JP PO,PATGO1	;D=C0H ?
EC27	16 01		LD D,01H	
EC29	3A EBF7	PATGO1:	LD A,(COLOR)	
EC2C	A2		AND D	;D=40H,80H or 01H
EC2D	C4 EC3D		CALL NZ,PATW	
;				
EC30	2A EBF5		LD HL,(PATB)	
EC33	01 4000		LD BC,4000H	
EC36	09		ADD HL,BC	
EC37	D8		RET C	
EC38	22 EBF5		LD (PATB),HL	
EC3B	18 DB		JR PATGO	
;.....				
EC3D	7E	PATW:	LD A,(HL)	
EC3E	08		EX AF,AF'	;A' HOLDS XCOUNT
EC3F	23		INC HL	
EC40	7E		LD A,(HL)	
EC41	23		INC HL	;HL=PAT-ADDR
EC42	D9		EXX	
EC43	57		LD D,A	;D'=YCOUNT
EC44	D9		EXX	
;				
EC45	C5	YLOPWB:	PUSH BC	;SAVE GPR-ADDR
EC46	08		EX AF,AF'	
EC47	57		LD D,A	;D=A'=XCOUNT
EC48	08		EX AF,AF'	;GET XCOUNT
;				
EC49	ED 78	XLOPWB:	IN A,(C)	
EC4B	AE		XOR (HL)	
EC4C	ED 79		OUT (C),A	
EC4E	23		INC HL	
EC4F	03		INC BC	;OUT 1 BYTE
;				



EC50	15	DEC	D	
EC51	C2 EC49	JP	NZ,XLOPWB	;SAME LINE GO ON
		;DOWN 1 LINE		
EC54	C1	POP	BC	;BACK ADDR
EC55	E5	PUSH	HL	;SAVE P-ADDR
EC56	3E 08	LD	A,08H	;VECTOR DOWN
EC58	80	ADD	A,B	;BC=ADDR, ADD HIGH BYTE
EC59	47	LD	B,A	
EC5A	E6 C0	AND	0C0H	;MASK 11000000
EC5C	BB	CP	E	;CHECK IN?
EC5D	CA EC66	JP	Z,DONEWB	;OK MASK
EC60	21 C050	PATCH1: LD	HL,0C050H	
EC63	09	;OR LD	HL,0C028H (WIDTH 40)	
EC64	44	ADD	HL,BC	
EC65	4D	LD	B,H	
		LD	C,L	
EC66	E1	DONEWB: POP	HL	
EC67	D9	EXX		
EC68	15	DEC	D	
EC69	D9	EXX		
EC6A	C2 EC45	JP	NZ,YLOPWB	
EC6D	C9	RET		
		;ADDR=4000H+(X1>>3)+((Y1 & 7)<<11) +(Y1>>3)*80		
		;ADDR=4000H+(BC/8) +((L AND 7)<<11)+(HL/8) *80		
		;HL=Y1,BC=X1,BREAKS HL,A,BC,DE		
		;return HL=addr,A=mask,D=count		
EC6E	7D	XYADDR: LD	A,L	;SAVE L
EC6F	CD EC99	CALL	DIV8	;HL=HL/8
EC72	54	LD	D,H	
EC73	5D	LD	E,L	;DE=HL
EC74	29	ADD	HL,HL	
EC75	29	ADD	HL,HL	;80=16*5
EC76	19	ADD	HL,DE	;HL=5
EC77	29	ADD	HL,HL	;10
EC78	29	ADD	HL,HL	;20
EC79	29	ADD	HL,HL	;40
EC7A	29	PATCH2: ADD	HL,HL	;80 ;HL=(HL/8)*80
		;OR NOP		(WIDTH 40)
EC7B	E6 07	AND	07H	;A=(L AND 7)
EC7D	87	ADD	A,A	
EC7E	87	ADD	A,A	
EC7F	87	ADD	A,A	;A=((L AND 7)<<3)
EC80	C6 40	ADD	A,040H	;ADD 4000H
EC82	57	LD	D,A	
EC83	1E 00	LD	E,00H	
EC85	19	;DE=4000H+((L AND 7)<<(3+8))		
		ADD	HL,DE	;LAST 2 AND 1ST WERE DON
EC86	EB	EX	DE,HL	;HDE=HL (SAVE)
EC87	60	LD	H,B	
EC88	69	LD	L,C	;HL=BC
EC89	7D	LD	A,L	;FOR bit7-0(BC=X1)
EC8A	CD EC99	CALL	DIV8	;HL=HL/8
EC8D	19	ADD	HL,DE	;BADDR DONE
		;NOW HL= BEGIN ADDR		
EC8E	E6 07	AND	07H	;CALC bit7-0
EC90	47	LD	B,A	
EC91	57	LD	D,A	
EC92	3E 80	LD	A,080H	
EC94	C8	RET	Z	;bit7 (<-AND 07H)
EC95	0F	SFLP: RRCA		
EC96	10 FD	DJNZ	SFLP	
EC98	C9	RET		
		;Now Acc has mask of 1 dot,		



```

;HL has result (address),
;D has mask count (0-7).
;
EC99    CB 3C
EC9B    CB 1D
EC9D    CB 3C
EC9F    CB 1D
ECA1    CB 3C
ECA3    CB 1D
ECA5    C9
;
DIV8:   SRL    H
        RR     L
        SRL    H
        RR     L
        SRL    H
        RR     L
        RET
;
;DEFAULT MOUSE CURSOR
ARW8:   DW     ARW80,ARW81 ;BIT SHIFTED PATTERN
        DW     ARW82,ARW83
        DW     ARW84,ARW85
        DW     ARW86,ARW87
;
ARW80:  DB     3,10
        DB     0FFH,0FCH,000H
        DB     0FFH,0F0H,000H
        DB     0FFH,0C0H,000H
        DB     0FFH,0F0H,000H
        DB     0FFH,0FCH,000H
        DB     0F3H,0FFH,000H
        DB     0C0H,0FFH,0C0H
        DB     000H,03FH,0F0H
        DB     000H,00FH,0C0H
        DB     000H,003H,000H
;
ARW81:  DB     3,10
        DB     07FH,0FEH,000H
        DB     07FH,0F8H,000H
        DB     07FH,0E0H,000H
        DB     07FH,0F8H,000H
        DB     07FH,0FEH,000H
        DB     079H,0FFH,080H
        DB     060H,07FH,0E0H
        DB     000H,01FH,0F8H
        DB     000H,007H,0E0H
        DB     000H,001H,080H
;
ARW82:  DB     3,10
        DB     03FH,0FFH,000H
        DB     03FH,0FCH,000H
        DB     03FH,0F0H,000H
        DB     03FH,0FCH,000H
        DB     03FH,0FFH,000H
        DB     03CH,0FFH,0C0H
        DB     030H,03FH,0F0H
        DB     000H,00FH,0FCH
        DB     000H,003H,0F0H
        DB     000H,000H,0C0H
;
ARW83:  DB     3,10
        DB     01FH,0FFH,080H
        DB     01FH,0FEH,000H
        DB     01FH,0F8H,000H
        DB     01FH,0FEH,000H
        DB     01FH,0FFH,080H
        DB     01EH,07FH,0E0H
        DB     018H,01FH,0F8H
        DB     000H,007H,0FEH
        DB     000H,001H,0F8H
        DB     000H,000H,060H
;
ARW84:  DB     3,10
        DB     00FH,0FFH,0C0H
        DB     00FH,0FFH,000H
        DB     00FH,0FCH,000H
        DB     00FH,0FFH,000H
        DB     00FH,0FFH,0C0H
        DB     00FH,03FH,0F0H
        DB     00CH,00FH,0FCH
        DB     000H,003H,0FFH
        DB     000H,000H,0FCH
        DB     000H,000H,030H
;
ARW85:  DB     4,10
        DB     007H,0FFH,0E0H,000H
        DB     007H,0FFH,080H,000H

```

S

ECA6	ECB6 ECD6
ECAA	ECF6 ED16
ECAE	ED36 ED56
ECB2	ED80 EDAA
ECB6	03 0A
ECB8	FF FC 00
ECBB	FF F0 00
ECBE	FF C0 00
ECC1	FF F0 00
ECC4	FF FC 00
ECC7	F3 FF 00
ECCA	C0 FF C0
ECCD	00 3F F0
ECD0	00 0F C0
ECD3	00 03 00
ECD6	03 0A
ECD8	7F FE 00
ECDB	7F F8 00
ECDE	7F E0 00
ECE1	7F F8 00
ECE4	7F FE 00
ECE7	79 FF 80
ECEA	60 7F E0
ECED	00 1F F8
ECF0	00 07 E0
ECF3	00 01 80
ECF6	03 0A
ECF8	3F FF 00
ECFB	3F FC 00
ECFE	3F F0 00
ED01	3F FC 00
ED04	3F FF 00
ED07	3C FF C0
ED0A	30 3F F0
ED0D	00 0F FC
ED10	00 03 F0
ED13	00 00 C0
ED16	03 0A
ED18	1F FF 80
ED1B	1F FE 00
ED1E	1F F8 00
ED21	1F FE 00
ED24	1F FF 80
ED27	1E 7F E0
ED2A	18 1F F8
ED2D	00 07 FE
ED30	00 01 F8
ED33	00 00 60
ED36	03 0A
ED38	0F FF C0
ED3B	0F FF 00
ED3E	0F FC 00
ED41	0F FF 00
ED44	0F FF C0
ED47	0F 3F F0
ED4A	0C 0F FC
ED4D	00 03 FF
ED50	00 00 FC
ED53	00 00 30
ED56	04 0A
ED58	07 FF E0 00
ED5C	07 FF 80 00



ED60	07 FE 00 00	DB	007H,0FEH,000H,000H
ED64	07 FF 80 00	DB	007H,0FFH,080H,000H
ED68	07 FF E0 00	DB	007H,0FFH,0E0H,000H
ED6C	07 9F F8 00	DB	007H,09FH,0F8H,000H
ED70	06 07 FE 00	DB	006H,007H,0FEH,000H
ED74	00 01 FF 80	DB	000H,001H,0FFH,080H
ED78	00 00 7E 00	DB	000H,000H,07EH,000H
ED7C	00 00 18 00	DB	000H,000H,018H,000H
;			
ED80	04 0A	ARW86: DB	4,10
ED82	03 FF F0 00	DB	003H,0FFH,0F0H,000H
ED86	03 FF C0 00	DB	003H,0FFH,0C0H,000H
ED8A	03 FF 00 00	DB	003H,0FFH,000H,000H
ED8E	03 FF C0 00	DB	003H,0FFH,0C0H,000H
ED92	03 FF F0 00	DB	003H,0FFH,0F0H,000H
ED96	03 CF FC 00	DB	003H,0CFH,0FCH,000H
ED9A	03 03 FF 00	DB	003H,003H,0FFH,000H
ED9E	00 00 FF C0	DB	000H,000H,0FFH,0C0H
EDA2	00 00 3F 00	DB	000H,000H,03FH,000H
EDA6	00 00 0C 00	DB	000H,000H,00CH,000H
;			
EDAA	04 0A	ARW87: DB	4,10
EDAC	01 FF F8 00	DB	001H,0FFH,0F8H,000H
EDB0	01 FF E0 00	DB	001H,0FFH,0E0H,000H
EDB4	01 FF 80 00	DB	001H,0FFH,080H,000H
EDB8	01 FF E0 00	DB	001H,0FFH,0E0H,000H
EDBC	01 FF F8 00	DB	001H,0FFH,0F8H,000H
EDC0	01 E7 FE 00	DB	001H,0E7H,0FEH,000H
EDC4	01 81 FF 80	DB	001H,081H,0FFH,080H
EDC8	00 00 7F E0	DB	000H,000H,07FH,0E0H
EDCC	00 00 1F 80	DB	000H,000H,01FH,080H
EDD0	00 00 06 00	DB	000H,000H,006H,000H
;			
EDD4	EDE4 EDFA	ARW4: DW	ARW40,ARW41 ;BIT SHIFTED PATTERN
S			
EDD8	EE10 EE26	DW	ARW42,ARW43
EDDC	EE3C EE52	DW	ARW44,ARW45
EDE0	EE68 EE7E	DW	ARW46,ARW47
;			
EDE4	02 0A	ARW40: DB	2,10
EDE6	FE 00	DB	0FEH,000H
EDE8	FC 00	DB	0FCH,000H
EDEA	F8 00	DB	0F8H,000H
EDEC	FC 00	DB	0FCH,000H
EDEE	FE 00	DB	0FEH,000H
EDF0	DF 00	DB	0DFH,000H
EDF2	8F 80	DB	08FH,080H
EDF4	07 C0	DB	007H,0C0H
EDF6	03 80	DB	003H,080H
EDF8	01 00	DB	001H,000H
;			
EDFA	02 0A	ARW41: DB	2,10
EDFC	7F 00	DB	07FH,000H
EDFE	7E 00	DB	07EH,000H
EE00	7C 00	DB	07CH,000H
EE02	7E 00	DB	07EH,000H
EE04	7F 00	DB	07FH,000H
EE06	6F 80	DB	06FH,080H
EE08	47 C0	DB	047H,0C0H
EE0A	03 E0	DB	003H,0E0H
EE0C	01 C0	DB	001H,0C0H
EE0E	00 80	DB	000H,080H
;			
EE10	02 0A	ARW42: DB	2,10
EE12	3F 80	DB	03FH,080H
EE14	3F 00	DB	03FH,000H
EE16	3E 00	DB	03EH,000H
EE18	3F 00	DB	03FH,000H
EE1A	3F 80	DB	03FH,080H
EE1C	37 C0	DB	037H,0C0H
EE1E	23 E0	DB	023H,0E0H
EE20	01 F0	DB	001H,0F0H
EE22	00 E0	DB	000H,0E0H
EE24	00 40	DB	000H,040H
;			
EE26	02 0A	ARW43: DB	2,10
EE28	1F C0	DB	01FH,0C0H
EE2A	1F 80	DB	01FH,080H
EE2C	1F 00	DB	01FH,000H
EE2E	1F 80	DB	01FH,080H
EE30	1F C0	DB	01FH,0C0H



EE32	1B E0		DB	01BH,0E0H
EE34	11 F0		DB	011H,0F0H
EE36	00 F8		DB	000H,0F8H
EE38	00 70		DB	000H,070H
EE3A	00 20		DB	000H,020H
;				
EE3C	02 0A	ARW44:	DB	2,10
EE3E	0F E0		DB	00FH,0E0H
EE40	0F C0		DB	00FH,0C0H
EE42	0F 80		DB	00FH,080H
EE44	0F C0		DB	00FH,0C0H
EE46	0F E0		DB	00FH,0E0H
EE48	0D F0		DB	00DH,0F0H
EE4A	08 F8		DB	008H,0F8H
EE4C	00 7C		DB	000H,07CH
EE4E	00 38		DB	000H,038H
EE50	00 10		DB	000H,010H
;				
EE52	02 0A	ARW45:	DB	2,10
EE54	07 F0		DB	007H,0F0H
EE56	07 E0		DB	007H,0E0H
EE58	07 C0		DB	007H,0C0H
EE5A	07 E0		DB	007H,0E0H
EE5C	07 F0		DB	007H,0F0H
EE5E	06 F8		DB	006H,0F8H
EE60	04 7C		DB	004H,07CH
EE62	00 3E		DB	000H,03EH
EE64	00 1C		DB	000H,01CH
EE66	00 08		DB	000H,008H
;				
EE68	02 0A	ARW46:	DB	2,10
EE6A	03 F8		DB	003H,0F8H
EE6C	03 F0		DB	003H,0F0H
EE6E	03 E0		DB	003H,0E0H
EE70	03 F0		DB	003H,0F0H
EE72	03 F8		DB	003H,0F8H
EE74	03 7C		DB	003H,07CH
EE76	02 3E		DB	002H,03EH
EE78	00 1F		DB	000H,01FH
EE7A	00 0E		DB	000H,00EH
EE7C	00 04		DB	000H,004H
;				
EE7E	03 0A	ARW47:	DB	3,10
EE80	01 FC 00		DB	001H,0FCH,000H
EE83	01 F8 00		DB	001H,0F8H,000H
EE86	01 F0 00		DB	001H,0F0H,000H
EE89	01 F8 00		DB	001H,0F8H,000H
EE8C	01 FC 00		DB	001H,0FCH,000H
EE8F	01 BE 00		DB	001H,0BEH,000H
EE92	01 1F 00		DB	001H,01FH,000H
EE95	00 0F 80		DB	000H,00FH,080H
EE98	00 07 00		DB	000H,007H,000H
EE9B	00 02 00		DB	000H,002H,000H
;				
END				

リスト 6-3 マウスイバ(リスト 6-2)のダンプリスト, もしくは  
"MOUSE.OBJ"でセーブしてねとあの子が言った

EA00	EB	5E	23	56	EB	22	29	EB	:	E3
EA08	11	31	EB	01	18	00	ED	B0	:	E3
EA10	3E	50	32	61	EC	3E	29	32	:	A6
EA18	7A	EC	3E	28	32	AC	EA	21	:	B5
EA20	A6	EC	22	2F	EB	3A	48	EB	:	3B
EA28	0F	38	17	47	3E	28	32	61	:	9E
EA30	EC	3E	00	32	7A	EC	3E	18	:	18
EA38	32	AC	EA	21	D4	ED	22	2F	:	FB
EA40	EB	78	0F	30	04	CD	D9	EB	:	37
EA48	C9	2A	31	EB	22	2B	EB	2A	:	71
EA50	33	EB	22	2D	EB	CD	DC	EA	:	EB
EA58	E6	03	28	05	3E	02	C3	16	:	2F
EA60	EB	CD	A5	EB	CD	D9	EB	FD	:	D6
EA68	21	39	EB	CD	49	EB	3A	39	:	B9
EA70	EB	F5	FD	21	39	EB	CD	49	:	38
EA78	EB	F1	B7	20	11	21	35	EB	:	05
-----										
SUM:	36	55	6F	EF	47	DE	8D	00	:	1738
EA80	3A	39	EB	E6	03	06	04	BE	:	0F
EA88	23	CA	11	EB	10	F9	21	3A	:	4D
EA90	EB	7E	23	B6	28	D8	3A	3A	:	B6
EA98	EB	CD	0A	EB	2A	31	EB	19	:	0C
EAA0	22	2B	EB	3A	3B	EB	CD	0A	:	6F
EAA8	EB	3E	01	BB	28	04	CB	2A	:	06
EAB0	CB	1B	2A	33	EB	19	22	2D	:	96
EAB8	EB	CD	DC	EA	FE	03	28	AE	:	55
EAC0	F5	CD	D9	EB	F1	0F	38	06	:	C4
EAC8	2A	2B	EB	22	31	EB	0F	38	:	C5
EAD0	06	2A	2D	EB	22	33	EB	CD	:	55



```

EAD8 D9 EB 18 92 AF ED 5B 44 : A9
EAE0 EB 2A 2D EB B7 ED 52 38 : 5B
EAE8 09 ED 5B 2D EB 2A 46 EB : C4
EAF0 ED 52 17 ED 5B 40 EB 2A : F3
EAF8 2B EB B7 ED 52 38 09 ED : 3A
-----
SUM: 00 00 7A 00 F3 BC 45 E3 B461

EB00 5B 2B EB 2A 42 EB ED 52 : 07
EB08 17 C9 5F 16 00 07 D0 15 : 41
EB10 C9 AF 18 02 3E 01 32 3C : 3F
EB18 EB CD D9 EB ED 5B 29 EB : D8
EB20 21 31 EB 01 18 00 ED B0 : F3
EB28 C9 1F 37 CD FC 0C E6 A3 : 7D
EB30 47 E6 80 78 CA 32 1F 32 : 72
EB38 BE 3E 06 80 22 BF 3E E6 : 87
EB40 10 C4 A9 04 3A EA 3C 87 : 68
EB48 3E 01 93 1F 3E 05 ED 79 : 9A
EB50 AF ED 79 16 80 15 20 FD : DD
EB58 3E 05 ED 79 3E 02 ED 79 : 4F
EB60 1E 05 FD 22 80 EB 16 03 : C6
EB68 CD 82 EB 38 09 FD 77 00 : EF
EB70 FD 23 15 20 F3 C9 FD 2A : 38
EB78 80 EB 1D 20 E9 C3 14 EB : 53
-----
SUM: B8 30 9F 3F 08 C5 1C 87 0616

EB80 1C E5 01 93 1F 21 00 00 : D5
EB88 2B 7C B5 28 0D AF ED 79 : A6
EB90 ED 78 1F 30 F3 0B ED 78 : 17
EB98 B7 C9 D5 CD A5 EB D1 2B : AE
EBA0 7C B5 20 FB C9 F3 CD AE : 83
EBA8 EB CD BA EB FB C9 01 A2 : C4
EBB0 1F 3E 47 ED 79 3E 1A ED : 4F
EBB8 79 C9 01 93 1F 21 CA EB : CB
EBC0 16 0F 7E 23 ED 79 15 20 : 61
EBC8 F9 C9 18 01 00 02 70 04 : 51
EBD0 44 05 00 06 00 07 00 03 : 59
EBD8 C1 ED 4B 31 EB 2A 33 EB : 5D
EBE0 ED 5B 3D EB 7A B3 20 04 : C1
EBE8 ED 5B 2F EB 3A 3F EB CD : 93
EBF0 F8 EB C9 3D FE 58 C2 00 : 01
EBF8 ED 53 F3 EB 0F 0F 32 F7 : 65
-----
SUM: BD E9 D5 77 B9 E6 14 1E 3C14

EC00 EB CD 6E EC 22 F5 EB 6A : 7E
EC08 26 00 29 ED 5B F3 EB 19 : 8E
EC10 11 F3 EB 01 02 00 ED B0 : 8F
EC18 2A F3 EB ED 4B F5 EB 78 : 98
EC20 E6 C0 5F 57 E2 29 EC 16 : 69
EC28 01 3A F7 EB A2 C4 3D EC : AC
EC30 2A F5 EB 01 00 40 09 D8 : 2C
EC38 22 F5 EB 18 DB 7E 08 23 : 9E
EC40 7E 23 D9 57 D9 C5 08 57 : CE
EC48 08 ED 78 AE ED 79 23 03 : A7
EC50 15 C2 49 EC C1 E5 3E 08 : F8
EC58 80 47 E6 C0 BB CA 66 EC : 44
EC60 21 50 C0 09 44 4D E1 D9 : 85
EC68 15 D9 C2 45 EC C9 7D CD : F4
EC70 99 EC 54 5D 29 29 19 29 : CA
EC78 29 29 29 E6 07 87 87 87 : FD
-----
SUM: 92 EE 18 64 CB 3B B5 4C 15F3

EC80 C6 40 57 1E 00 19 EB 60 : DF
EC88 69 7D CD 99 EC 19 E6 07 : 3E
EC90 47 57 3E 80 C8 0F 10 FD : 40
EC98 C9 CB 3C CB 1D CB 3C CB : 8A

```

```

ECA0 1D CB 3C CB 1D C9 B6 EC : 77
ECA8 D6 EC F6 EC 16 ED 36 ED : CA
ECB0 56 ED 80 ED AA ED 03 0A : 54
ECB8 FF FC 00 FF F0 00 FF C0 : A9
ECC0 00 FF F0 00 FF FC 00 F3 : DD
ECC8 FF 00 C0 FF C0 00 3F F0 : AD
ECD0 00 0F C0 00 03 00 03 0A : DF
ECD8 7F FE 00 7F F8 00 7F E0 : 53
ECE0 00 7F F8 00 7F FE 00 79 : 6D
ECE8 FF 80 60 7F E0 00 1F F8 : 55
ECF0 00 07 E0 00 01 80 03 0A : 75
ECF8 3F FF 00 3F FC 00 3F F0 : A8
-----
SUM: 43 90 F8 E1 B4 29 2D 0A FF0E

ED00 00 3F FC 00 3F FF 00 3C : B5
ED08 FF C0 30 3F F0 00 0F FC : 29
ED10 00 03 F0 00 00 C0 03 0A : C0
ED18 1F FF 80 1F FE 00 1F F8 : D2
ED20 00 1F FE 00 1F FF 80 1E : D9
ED28 7F E0 18 1F F8 00 07 FE : 93
ED30 00 01 F8 00 00 60 03 0A : 66
ED38 0F FF C0 0F FF 00 0F FC : E7
ED40 00 0F FF 00 0F FF C0 0F : EB
ED48 3F F0 0C 0F FC 00 03 FF : 48
ED50 00 00 FC 00 00 30 04 0A : 3A
ED58 07 FF E0 00 07 FF 80 00 : 6C
ED60 07 FE 00 00 07 FF 80 00 : 8B
ED68 07 FF E0 00 07 9F F8 00 : 84
ED70 06 07 FE 00 00 01 FF 80 : 8B
ED78 00 00 7E 00 00 00 18 00 : 96
-----
SUM: 06 02 AD 9B 63 EB A0 F4 A45C

ED80 04 0A 03 FF F0 00 03 FF : 02
ED88 C0 00 03 FF 00 00 03 FF : C4
ED90 C0 00 03 FF F0 00 03 CF : 84
ED98 FC 00 03 03 FF 00 00 00 : 01
EDA0 FF C0 00 00 3F 00 00 00 : FE
EDA8 0C 00 04 0A 01 FF F8 00 : 12
EDB0 01 FF E0 00 01 FF 80 00 : 60
EDB8 01 FF E0 00 01 FF F8 00 : D8
EDC0 01 E7 FE 00 01 81 FF 80 : E7
EDC8 00 00 7F E0 00 00 1F 80 : FE
EDD0 00 00 06 00 E4 ED FA ED : BE
EDD8 10 EE 26 EE 3C EE 52 EE : 7C
EDE0 68 EE 7E EE 02 0A FE 00 : CC
EDE8 FC 00 F8 00 FC 00 FE 00 : EE
EDF0 DF 00 8F 80 07 C0 03 80 : 38
EDF8 01 00 02 0A 7F 00 7E 00 : 0A
-----
SUM: E2 8B 80 50 C6 23 60 28 E48B

EE00 7C 00 7E 00 7F 00 6F 80 : 68
EE08 47 C0 03 E0 01 C0 00 80 : 2B
EE10 02 0A 3F 80 3F 00 3E 00 : 48
EE18 3F 00 3F 80 37 C0 23 E0 : F8
EE20 01 F0 00 E0 00 40 02 0A : 1D
EE28 1F C0 1F 80 1F 00 1F 80 : 3C
EE30 1F C0 1B E0 11 F0 00 F8 : D3
EE38 00 70 00 20 02 0A 0F E0 : 8B
EE40 0F C0 0F 80 0F C0 0F E0 : 1C
EE48 0D F0 08 F8 00 7C 00 38 : B1
EE50 00 10 02 0A 07 F0 07 E0 : FA
EE58 07 C0 07 E0 07 F0 06 F8 : A3
EE60 04 7C 00 3E 00 1C 00 08 : E2
EE68 02 0A 03 F8 03 F0 03 E0 : DD
EE70 03 F0 03 F8 03 7C 02 3E : AD
EE78 00 1F 00 0E 00 04 03 0A : 3E

```



```
-----
SUM: 6F BF 5F DE 4B 62 24 62 B9D0
EE80 01 FC 00 01 F8 00 01 F0 : E7
EE88 00 01 F8 00 01 FC 00 01 : F7
```

```
EE90 BE 00 01 1F 00 00 0F 80 : 6D
EE98 00 07 00 00 02 00 : 09
-----
SUM: BF 04 F9 20 FB FC 10 71 39E5
```

リスト 6-4 マウスが舞うす、もしくは早くパターンエディタになりたい

```
100 CLEAR &HE000
110 IF MEM$(&HEA00,4)<>HEXCHR$("EB 5E 23 56") THEN LOADM "MOUSE.OBJ"
120 DEFUSR0=&HEA00
130 X=100:Y=100
140 MEM$(&HE000,4)=MKI$(X)+MKI$(Y) : 'X Y
150 'MEM$(&HE000,4)=HEXCHR$("64 00 64 00") : 'X Y
160 MEM$(&HE004,4)=HEXCHR$("01 02 03 04") : 'BUTTON CONDITION
170 MEM$(&HE008,4)=HEXCHR$("00 00 00 00") : 'MOUSE DATA
180 MEM$(&HE00C,3)=HEXCHR$("00 00 04") : 'MOUSE CURSOR
190 X0=40:X1=240:Y0=24:Y1=160
200 MEM$(&HE00F,8)=MKI$(X0)+MKI$(X1-1)+MKI$(Y0)+MKI$(Y1-1)
210 'MEM$(&HE00F,8)=HEXCHR$("28 00 EF 00 18 00 9F 00") : 'AREA
220 MEM$(&HE017,1)=HEXCHR$("00") : 'MODE
230 WIDTH40
240 '
250 'MEM$(&HE00C,3)=HEXCHR$("80 D0 07") : 'ト スト CURSOR カ' ' ' ニ ナル
260 'MEM$(&HE080,16)=HEXCHR$("90 D0 93 D0 96 D0 99 D0 9C D0 9F D0 A2 D0 A5 D0")
270 'MEM$(&HE090,16)=HEXCHR$("01 01 80 01 01 40 01 01 20 01 01 10 01 01 08 01")
280 'MEM$(&HE0A0,16)=HEXCHR$("01 04 01 01 02 01 01 01")
290 CLS 4:INIT
300 LINE(X0-1,Y0-1)-(X1,Y1),PSET,7,B
310 LINE((X0-1)/8,(Y0-1)/8)-((X1-8)/8,(Y1-8)/8),CHR$(225),BF
320 '
330 D$=USR0(CHR$(&H0,&HE0))
340 '
350 BT=PEEK(&HE008) AND 3
360 IF BT=3 THEN WIDTH80:END
370 IF BT=1 THEN C=224:GOSUB "PSET"
380 IF BT=2 THEN C=225:GOSUB "PSET"
390 GOTO 330
400 '
410 LABEL"PSET"
420 X=CVI(MEM$(&HE000,2)):X=INT(X/8)
430 Y=CVI(MEM$(&HE002,2)):Y=INT(Y/8)
440 PSET(X+239+1,Y+159+1,(225-C)*7)
450 LOCATEX,Y:PRINTCHR$(C):RETURN
```

打ち込んだなら、

SAVEM "MOUSE. OBJ", &HEA00, &HEE9D

でセーブしていただきたい（テープでも可だけど、はっきり言ってマウスより先にディスクを買うべきである）。次にリスト 6-4 を RUN する（おっと、turbo BASIC の場合は事前に「KMODE 0」を実行しておくこと）。するとディスクが回り、「MOUSE. OBJ」がロードされた後、何やらパターンエディタもどきの画面になるであろう。セコイことにモノクロで、気分の良いことにセーブ機能などは一切付いていない。なぜならばこのプログラムは、私が苦心してデザインしたマウスカーソル（本当はただの左上向きの矢印だけど）が、画面を華麗に舞う姿を楽しむためだけ、のものだからである。左側のボタンを押すと白丸が、右側のボタンを押すと中抜き丸が表示されるであろう。終わりたいときは両方のボタンを同時に押すのであるが、人間の指はどうしても左右のボタンを押すのに時間差を生じてしまうらしく、なかなか難しいはずである。いざとなったら、**SHIFT** + **BREAK** をちょっと押した後、どちらか片方のボタンで BREAK する。

まずはリスト 6-4 の説明、すなわちリスト 6-3 の機械語ルーチンの使い方である。140～220 行はパラメータの指定である。中身は、



① マウスカーソルの位置 (X, Y の順)

2 バイトの値で指定する。MKI\$の使い方に注意。合計 4 バイトである。

② どのボタンが押されたらリターンするか

これは 4 とおり指定できる。

00<sub>H</sub> = ボタンが押されてなかったら RET

01<sub>H</sub> = 左側ボタンだけが押されたら RET

02<sub>H</sub> = 右側ボタンだけが押されたら RET

03<sub>H</sub> = 両側とも押されたら RET

04<sub>H</sub> = 無効

リスト 6-4 では、とにかく何かボタンが押されたら RET するように 01<sub>H</sub>, 02<sub>H</sub>, 03<sub>H</sub> の 3 とおりを指定している。04<sub>H</sub> は「詰め物」のよーなものである。合計 4 バイト。

③ マウスデータ

最初の 3 バイトには、最後にマウスから読み出された生のデータが入っている。4 バイト目はエラーコードである。

00<sub>H</sub> → 正常

01<sub>H</sub> → マウスがつながっていないなど

02<sub>H</sub> → マウスカーソルの位置が、最初からウィンドウの外側にある。

合計 4 バイト。

④ マウスカーソルの形状、色の指定

最初の 2 バイトはカーソル用のグラフィックのデータが格納されているアドレスである。0000<sub>H</sub> を指定すると、デフォルトの左上向き矢印になる。3 バイト目は色の指定。リスト 6-4 では緑になっている。合計 3 バイト。

⑤ ウィンドウ（移動範囲）の指定

X 方向の下限值, X 方向の上限値, Y 方向の下限值, Y 方向の上限値, の順に指定する。それぞれ 2 バイトの値である。リスト 5 では、マウスカーソルの左上端の位置が、(40, 24) - (240 - 1, 160 - 1) の囲いの中から出ないようにしている。合計 8 バイト。

⑥ モードの指定

第 0 ビットは WIDTH の指定。0 なら 40 字モード, 1 なら 80 字モードとなる。第 1 ビットはマウス動作をするか、ただのグラフィック描画をするかのフラグである。0 ならマウス動作, 1 なら④で指定されたマウスカーソルを XOR モードで表示するだけですぐ RET する。合計 1 バイト。

ま、大体以上なわけである。よって、正しいリスト 6-3 の使い方は、①～⑥のパラメータを指定して、330 行のようにそのパラメータの先頭番地をもって、USR 命令である。その後、ボタンの状態を知りたいのなら③の先頭 1 バイト（リスト 6-4 では E008<sub>H</sub> 番地）を PEEK して AND 3, マウスカーソルの位置を知りたいなら、420 行, 430 行のようにすればよい。その後 8 で割っているのはキャラクタの位置に変換しているのだから勘違いはしないように。

それではたちまちリスト 6-2 の説明に入る。最初の EA0F<sub>H</sub> まではパラメータの転送など



である。扱いやすいように EB31<sub>H</sub>からの 24 バイトに転送して使うのである (RET するときには、元に転送し直す)。

次から EA41<sub>H</sub>までは、40/80 字モードの切り換えで、プログラムを直接書き換えるという最低のテクニックを使っている。パラメータの⑥の第 1 ビットが 1 ならば、EA45<sub>H</sub>番地に来てたちまち RET、さもなくば EA49<sub>H</sub>からのマウスドライバに入る。

後は詳しい説明はしないで、サブルーチンの説明にとどめておく。

CIN=EADC<sub>H</sub>は、(X0, Y0) がウィンドウの中に入っているかどうかの判定ルーチン。ウィンドウの中に入っていなければ A レジスタのビットを立てる (ビット 1 は Y, ビット 0 は X)。

XSIGN=EB0A<sub>H</sub>は 8 ビットの符号付きの E レジスタを 16 ビット値にして DE レジスタに入れるもの。

MSIN =EB49<sub>H</sub>はマウスからの入力

SETMS =EBA5<sub>H</sub>はマウスのイニシャライズ

PAT =EBF8<sub>H</sub>はマウスカーソルの描画

XYADDR =EC6E<sub>H</sub>は座標(X, Y)から G-RAM のアドレスを計算するルーチン

そして、ECA6<sub>H</sub>以降はグラフィックデータである。グラフィックデータは描画をスムーズにするため、あらかじめ 1 ビットずつシフトしたデータを 8 とおり用意しておく方法を採用している。メモリをけちって、実行時にいちいちシフトさせるとマウスの操作性がガクンと落ちるのである。40 字モードと 80 字モードの 2 とおりのデータで 500 バイト近くある。清く正しいプログラムでは、起動直前にシフトしたデータを作るべきなのであるが、しっかり手抜きをしているわけである。

では、「マウスを生かすも殺すもソフトしだい」とつぶやきつつ、この章を終わるのである。







第

7

章

通 信



通信だってするのである



## ■第7章

# 通信だってするのである……………

この章では前章の舌の根も乾かぬうちに通信をするのである。具体的には CZ-8BM2 (RS-232C・マウスボード) 用のコントロールプログラムなのである。プログラムは turbo BASIC の RS-232C 関係の命令と同等のことができるようにすることを目的として作った。よって turbo ユーザーにとっては、基本的にお呼びでない。しかしプログラム中の SIO, CTC のアドレスをずらし、さらに CZ-8FB01 を使うか(もしくは CZ-8FB02 であっても割り込みベクトルのつじつまを合わせる) すれば turbo でも使えることは使えるから、「BASIC に依存しない通信プログラム」を作る際の参考ぐらいにはなるであろう。

で、実は CZ-8BM2 にはカセットに入ったアプリケーションプログラム(機械語)が付いてくるのであるが、どうやらこの章のプログラムはそれよりはましなものになったようである。

ところで通信であるが、私の持論では「通信＝不毛」ということになっている。誰が何と言っても現時点では(1987年である)NTTが儲かるだけなのだ。モデムに必要なモジュラージャックを取り付ける工事だけで何千円もボるような(資格のない人が取り付けるのは禁止されている)。また、日本ではぼちぼちと通信速度が 1200 bps, さらにそれ以上へと移行しつつあるようだが、はっきり言ってそれでもまだタコなのである。そして一番の問題は「パソコン通信なんかやって何が面白いの?」である。おそらく面白いのは CHAT だけであろう。後は PDS (パブリックドメインソフト) が手に入るなどというメリットもあるが、よーするに現状のパソコン通信は「それだけのもの」なのである。

もう少し言わせてもらおうと、私は CHAT が嫌いである。やってみれば楽しいのだろうが、それでも嫌いである。なぜかと言うと、文明社会の「ユガミ」みたいなものを感じてしまうからなのだ。これはあくまでも想像するだけなのだが、CHAT の情熱の嵐に巻き込まれたら最後なのではないだろうか。すなわち CHAT に命をかけているよーな人は、他人とのコミュニケーションにどっかこっか問題があるのではないかと思えるのである。面と向かって肉声で話すよりも、キーボードをパコパコ叩いて文字で話す方がいいなんて、なんか不自然だろう。

というわけで、この章のプログラムは、できればメディアコンパートなどに使い、くれぐれもパソコン通信などには使わないよーにお願いしたい。

## それではいきなり始める

X1 のユーザーは turbo BASIC のマニュアルを持っていないわけだから、表 7-1 (turboZ のマニュアルから引用) を参考にさせていただきたい。ただしプログラムは BASIC のコ



マンドを拡張するわけではないので、USR0～USR9 を使うことになっている。その際、文字列変数を渡すときには「その長さ」にちょっと気を付けなければいけないので、サンプルプログラムをよく見て使うように。なお、SAVE, SAVEM, LOAD, LOADM はないので、自分で(BASICなどで)プログラムを組むなどして解決するよーに。

表 7-1 通信パラメータの説明

通信パラメータには次のものがある。							
1) ボーレート							
2) パリティ							
3) データビット長							
4) ストップビット長							
5) 通信制御指定							
6) カナの表現方法指定							
7) CR, LFコードの送信処理							
8) CR, LFコードの受信処理							
9) 日本語文字列の表現方法指定 (本書のプログラムでは“N”しか使えない)							
10) エンドコード指定							
1) ボーレート							
0～6の数値によりボーレートを指定する。							
数 値	0	1	2	3	4	5	6
ボーレート (bps)	150	300	600	1200	2400	4800	9600
2) パリティ							
E: 偶数パリティチェックを使用する。							
O: 奇数パリティチェックを使用する。							
N: パリティチェックを使用しない。							
3) データビット長							
5: データビット長を5ビットとする。							
6: データビット長を6ビットとする。							
7: データビット長を7ビットとする。							
8: データビット長を8ビットとする。							
4) ストップビット長							
1: ストップビット長を1ビットとする。							
2: ストップビット長を1.5ビットとする。							
3: ストップビット長を2ビットとする。							
5) 通信制御指定							
X: XON/XOFFコードによる制御を行なう。							
R: RTS制御信号のON/OFFによる制御をする。							
Nまたは省略: 通信制御を行なわない。							
この通信制御とは、データ送受信時に受信側においてデータの受信処理が間に合わないとき、送信側に対して送信の一時停止を要求し受信できる状態になると送信再開を要求する方法のことをいう。							
“X”を指定すると、データとしてXOFFコード(&H13)を送信側へ送ることによって送信の一時停止を要求し、XONコード(&H11)により送信再開を要求する。							
“R”を指定すると、RTS信号線をOFFにすることによって送信の一時停止を要求し、ONにすることにより送信再開を要求する。							
6) カナの表現方法指定							
S: データビット長7ビットでカナの送受信ができる。							
Nまたは省略: データビット長7ビットでカナの送受信ができない。							
データビット長7ビットでカナの送受信はシフトインコードSI(&H0F)があると、それ以降のデータをカナとして処理し、シフトアウトコードSO(&H0E)があればそれ以降のデータを英数字として処理する。							
7) CR, LFコードの送信処理							
C: 復帰改行コードとしてCRコード(&H0D)を送信する。							
Lまたは省略: 復帰改行コードとしてCRコード(&H0D)+LFコード(&H0A)を送信する。							
8) CR, LFコードの受信処理							
C: CRコード(&H0D)の受信によって復帰+改行処理する。							
Lまたは省略: CRコード(&H0D)のみを受信するとそれはデータとして処理される。CRコード(&H0D)+LFコード(&H0A)を連続して受信すると、復帰+改行処理をする。							
9) 日本語文字列の表現方法指定							
J: JIS漢字コードを使用する。漢字インコードKI(&H1B4B)で日本語文字列の始まりを示し、漢字アウトコードKO(&H1B48)で日本語文字列の終わりを示す。							
Nまたは省略: シフトJIS漢字コードを使用する。(本書のプログラムでは“N”しか使えない)							
10) エンドコード指定							
データ転送の終了を判断するためのエンドコードを指定する。エンドコードとしてコントロールコード(&H00～&H1F)の中から任意の一つを選択できる。実際に指定するパラメータは、キャラクターコード &H40～&H5F に対応する文字を使用する(&H60～&H7Fも可)。たとえば、D(またはd)と指定するとエンドコードとしてコントロールD(&H04)が使用される。またこのパラメータを省略すると、エンドコードの送受信処理は行なわない。							
[注]							
●通信パラメータのうちボーレート、パリティ、データ長、ストップビット長は省略できないが、それ以降のパラメータは省略できる。ただし、途中のパラメータを省略して次のパラメータを指定することはできない。							
●漢字の送受信はデータ長8ビットのときに可能(シフトJISコードを使うことになる)。また、カタカナの送受信はSI/SOコードを使うことによってデータ長が7ビットのときでも可能となる(カナの表現方法指定がSであれば自動的にカナに変換する)。							



各USR関数について		USR5: PRINT	文字列+CR(+LF) を出力する
USR0: OPEN	オープンする (パラメータ先頭に O,I,C のいずれかを付加すること)	USR6: LOC	バッファ内のデータ数を返す
USR1: CLOSE	クローズする	USR7: EOF	エンドコードを入力したかどうかを返す
USR2: LINPUT	1行入力する (CRもしくはCR+LFまで)	USR8: PGETC1	生データを1文字入力する (カナの処理などはしない)
USR3: INPUTN	N個の文字を入力する (個数は引数の文字列の先頭のASCIIコード)	USR9: PPUTC1	生データを1文字出力する (カナの処理などはしない)
USR4: PRINTS	文字列を出力する	注) USR2, USR3は、返す文字列の先頭のASCIIコードで入力されたデータ数を示す。	

そしてリスト 7-1 = ソースリスト, リスト 7-2 = ダンプリスト (打ち込み確認には付録 A を参照のこと), リスト 7-3 = サンプルプログラム, リスト 7-4 = 簡単なターミナルソフトである (ただし使いものにはならないので, あくまでサンプルにすぎない)。リスト 7-1 の最初の方を見ると分かるだろうが, Z80 SIO, CTC のアドレスがそれぞれ 1F98<sub>H</sub> と 1FA8<sub>H</sub> となっている。これは CZ-8BM2 のアドレス設定スイッチが, 工場出荷時の 2-3 側になっている場合のものである。スイッチが 1-2 側もしくは turbo の場合には, それぞれ 1F90<sub>H</sub>, 1FA0<sub>H</sub> とすればよい。後は説明しても空しいのでやめておく。

リスト 7-1 通信プログラムのソースリスト

		.Z80	
		.PHASE	0EA00H
0330		QBREAK	EQU 0330H
1FCA		BBREAK	EQU 1FCAH
		;	
1F98		ZSIO	EQU 1F98H ;IF NOT,1F98H
1FA8		ZCTC	EQU 1FA8H ;IF NOT,1FA8H
		;	
005C		INTRSV	EQU 5CH ;INT VECTOR
0040		QMAX	EQU 64 ;QUE SIZE
0080		QMAXX	EQU QMAX+64 ;Q SIZE+ALPHA
		;	
000D		CR	EQU 0DH
000A		LF	EQU 0AH
000E		SO	EQU 0EH
000F		SI	EQU 0FH
0011		XON	EQU 11H
0013		XOFF	EQU 13H ;CTRL CODES
		;	
		BAU RATE	0-6
		PARITY	E,O,N
		DATA BIT LEN	5,6,7,8
		STOP BIT LEN	1,2,3
		CONTROLE	X,R,N,""
		KANA MODE	S,N,""
		CR,LF SEND	C,L,""
		CR,LF RECEIVE	C,L,""
		KANJI	J,N,"" ;DUMMY
		END CODE	'@'-'_ ' ;PARAMS
		;	
		JUMP TABLE	
EA00	C3 EA1E	JP	OPEN ;+0
		;	
EA03	C3 EB84	JP	CLOSE ;+3
		;	
		TX ENDCODE	
		;	
EA06	C3 ECDD	JP	LINPUT ;+6
		;	
EA09	C3 ED35	JP	INPUTN ;+9
		;	
		INPUT n CHAR	



EA0C	C3 ED68	;	JP	PRINTS	;	+12
EA0F	C3 ED56	;	TX STRING	JP	PRINT	;
						+15
			;	TX STRING + CR (+LF)		
EA12	C3 EB98	;	JP	LOC	;	+18
EA15	C3 EBB9	;	RETURN	BUFF SIZE		
			JP	EOF	;	+21
			;	EOF OR NOT		
EA18	C3 EC9A	;	JP	PGETC1	;	+24
EA1B	C3 ED90	;	RIMITIVE	GET CHAR		
			JP	PPUTC1	;	+27
			;	PRIMITIVE PUT CHAR		
EA1E	3A EE60	OPEN:	LD	A, (OPENF)		
EA21	B7		OR	A		
EA22	28 07		JR	Z, OPENGO		
		;				
EA24	D5	;	REOPEN	THEN CLOSE		
EA25	C5		PUSH	DE		
EA26	CD EB84		PUSH	BC		
EA29	C1		CALL	CLOSE		
EA2A	D1		POP	BC		
			POP	DE		
EA2B	3E FF	OPENGO:	LD	A, 0FFH		
EA2D	32 EE60		LD	(OPENF), A	;	SET FLAG
		;				
EA30	3D	OPENW:	DEC	A		
EA31	20 FD		JR	NZ, OPENW	;	WAIT
		;				
EA33	F3		DI			
EA34	21 EBC6		LD	HL, INTRSR		
EA37	22 005C		LD	(INTRSV), HL		
EA3A	FB		EI		;	SET VECTOR
		;				
EA3B	EB		EX	DE, HL		
EA3C	78		LD	A, B	;	COPY LEN
EA3D	B7		OR	A		
EA3E	CA EE26		JP	Z, ERR	;	NULL STR
EA41	05		DEC	B		
EA42	7E		LD	A, (HL)	;	1 CHAR
EA43	23		INC	HL		
EA44	FE 49		CP	'I'		
EA46	28 0B		JR	Z, OPENOK		
EA48	FE 4F		CP	'O'		
EA4A	28 07		JR	Z, OPENOK		
EA4C	FE 43		CP	'C'		
EA4E	28 03		JR	Z, OPENOK		
EA50	C3 EE26		JP	ERR		
		;				
EA53	32 EE4B	OPENOK:	LD	(IOC), A	;	I, O or C
EA56	78		LD	A, B	;	GET COUNT
EA57	D6 04		SUB	4		
EA59	DA EE26		JP	C, ERR		
EA5C	FE 07		CP	7		
EA5E	D2 EE26		JP	NC, ERR		
EA61	4F		LD	C, A	;	COUNTER
EA62	7E		LD	A, (HL)		
EA63	23		INC	HL		
		;				
EA64	D6 30	;	BAU RATE	0-6		
EA66	FE 07		SUB	'0'		
EA68	D2 EE26		CP	7		
EA6B	16 D0		JP	NC, ERR		
EA6D	D6 02		LD	D, 208		
EA6F	F5		SUB	2		
EA70	30 01		PUSH	AF		
EA72	AF		JR	NC, OPEN0		
EA73	B7		XOR	A		
EA74	28 05	OPEN0:	OR	A		
EA76	CB 3A		JR	Z, OPEN1		
EA78	3D		SRL	D	;	D=D/2
EA79	18 F8		DEC	A		
			JR	OPEN0		
		;				
EA7B	7A	OPEN1:	LD	A, D		
EA7C	32 EE2E		LD	(CTR), A		



136 試験に出る X1



EAEB	21	EE3F	LD	HL,PDFLT	
EAEE	11	EE45	LD	DE,PARAM	
EAFL	01	0006	LD	BC,6	
EAFL	ED	B0	LDIR		;COPY DEFAULT
EAFL	EB		EX	DE,HL	;DE'=PARAM
EAFL	D9		EXX		
EAFL	E1		POP	HL	
EAFL	C1		POP	BC	
EAFA	11	EE2F	LD	DE,PLIST	
EAFL	AF		XOR	A	;FLAG
EAFL	08		EX	AF,AF'	;A'=0
EAFL	79		LD	A,C	
EB00	B7		OR	A	
EB01	28	1F	JR	Z,FINIS	
EB03	FE	06	CP	6	
EB05	20	02	JR	NZ,OPEN6	
EB07	0D		DEC	C	;C=5
EB08	08		EX	AF,AF'	;A'<>0
EB09	46				
EB0A	23		OPEN6:	LD	B,(HL)
EB0B	1A			INC	HL
EB0C	13		OPEN7:	LD	A,(DE)
EB0D	B7			INC	DE
EB0E	CA	EE26		OR	A
EB11	B8			JP	Z,ERR
EB12	20	F7		CP	B
				JR	NZ,OPEN7
			;MATCH		
EB14	D9			EXX	
EB15	12			LD	(DE),A
EB16	D9			EXX	
EB17	1A		OPEN75:	LD	A,(DE)
EB18	13			INC	DE
EB19	B7			OR	A
EB1A	20	FB		JR	NZ,OPEN75
EB1C	D9		OPEN8:	EXX	
EB1D	13			INC	DE
EB1E	D9			EXX	
EB1F	0D			DEC	C
EB20	20	E7		JR	NZ,OPEN6
EB22	08		FINIS:	EX	AF,AF'
EB23	B7			OR	A
EB24	28	10		JR	Z,SETZ
EB26	7E			LD	A,(HL)
EB27	FE	40		CP	40H
EB29	DA	EE26		JP	C,ERR
EB2C	FE	80		CP	80H
EB2E	D2	EE26		JP	NC,ERR
EB31	E6	1F		AND	1FH
EB33	D9			EXX	
EB34	12			LD	(DE),A
EB35	D9			EXX	;STORE
EB36	F3		SETZ:	DI	;SET ZCTC,ZSIO
EB37	CD	EB6A		CALL	INITBF
EB3A	01	1FA9		LD	BC,ZCTC+1
EB3D	3E	47		LD	A,01000111B
EB3F	ED	79		OUT	(C),A
EB41	3A	EE2E		LD	A,(CTR)
EB44	ED	79		OUT	(C),A
					;T CONSTANT
EB46	21	EE4C		LD	HL,SIOA
EB49	01	1F99		LD	BC,ZSIO+1
EB4C	3E	0F		LD	A,15
EB4E	CD	EB63		CALL	SETSIO
EB51	21	EE5B		LD	HL,SIOB
EB54	01	1F9B		LD	BC,ZSIO+3
EB57	3E	05		LD	A,5
EB59	CD	EB63		CALL	SETSIO
EB5C	3E	FF		LD	A,0FFH
EB5E	32	EE60		LD	(OPENF),A
EB61	FB			EI	
EB62	C9			RET	
EB63	04		SETSIO:	INC	B
EB64	ED	A3		OUTI	
EB66	3D			DEC	A
EB67	20	FA		JR	NZ,SETSIO
EB69	C9			RET	



EB6A	AF	;	INITBF:	XOR	A	
EB6B	32 EE6A			LD	(QLEN),A	
EB6E	21 EE6B			LD	HL,Q0	
EB71	22 EE66			LD	(HEAD),HL	
EB74	22 EE68			LD	(TAIL),HL	
			;			
EB77	32 EE62			LD	(LASTX),A	
EB7A	32 EE63			LD	(XFLAG),A	
EB7D	32 EE64			LD	(TKIN),A	
EB80	32 EE65			LD	(RKIN),A	;RESET MODE
EB83	C9			RET		
			;			
EB84	AF		CLOSE:	XOR	A	
EB85	32 EE60			LD	(OPENF),A	
			;			
EB88	3A EE4B			LD	A,(IOC)	
EB8B	FE 4F			CP	'O'	
EB8D	C0			RET	NZ	
EB8E	3A EE4A			LD	A,(ECODE)	
EB91	FE 20			CP	20H	
EB93	C8			RET	Z	
EB94	CD EDCD			CALL	PPUTC	;TX ECODE
EB97	C9			RET		
			;			
EB98	3A EE6A		LOC:	LD	A,(QLEN)	
EB9B	77			LD	(HL),A	
EB9C	23			INC	HL	
EB9D	36 00			LD	(HL),0	;RET SIZE
EB9F	C9			RET		
			;			
EBA0	3A EE6A		;	BREAK	B,A,F	
EBA3	B7		QEOF:	LD	A,(QLEN)	
EBA4	C8			OR	A	
EBA5	3A EE4A			RET	Z	;EMPTY
EBA8	FE 20			LD	A,(ECODE)	
EBAA	C8			CP	20H	
EBAB	E5			RET	Z	;NO ECODE
EBAC	2A EE68			PUSH	HL	
EBAF	46			LD	HL,(TAIL)	
EBB0	E1			LD	B,(HL)	;NOZOKU
EBB1	B8			POP	HL	
EBB2	28 02			CP	B	
EBB4	AF			JR	Z,QEOF1	
EBB5	C9			XOR	A	;SET Z
EBB6	F6 FF		QEOF1:	RET		
EBB8	C9			OR	0FFH	;SET NZ
				RET		
			;			
EBB9	CD EBA0		EOF:	CALL	QEOF	
EBBC	11 0000			LD	DE,0	
EBBF	28 01			JR	Z,EOF1	
EBC1	1B			DEC	DE	
EBC2	73		EOF1:	LD	(HL),E	
EBC3	23			INC	HL	
EBC4	72			LD	(HL),D	
EBC5	C9			RET		
			;			
EBC6	F3		INTRSR:	DI		;INT ROUTINE
EBC7	F5			PUSH	AF	
EBC8	C5			PUSH	BC	
EBC9	D5			PUSH	DE	
EBCA	E5			PUSH	HL	
			;			
EBCB	01 1F99			LD	BC,ZSIO+1	
			;			
EBCE	3E 01		INTR1:	LD	A,1	
EBD0	ED 79			OUT	(C),A	;RR1
EBD2	ED 78			IN	A,(C)	
			;			
EBD4	0B			DEC	BC	
EBD5	ED 78			IN	A,(C)	
EBD7	2A EE66			LD	HL,(HEAD)	
EBDA	77			LD	(HL),A	
EBDB	23			INC	HL	
EBDC	E5			PUSH	HL	
EBDD	01 EEEB			LD	BC,Q1	
EBE0	B7			OR	A	
EBE1	ED 42			SBC	HL,BC	
EBE3	E1			POP	HL	
EBE4	20 03			JR	NZ,INTR2	



EBE6	21	EE6B			
EBE9	22	EE66			
EBEC	21	EE6A			
EBEF	34				
EBF0	FE	13			
EBF2	20	05			
EBF4	32	EE62			
EBF7	18	08			
EBF9	FE	11			
EBFB	20	04			
EBFD	AF				
EBFE	32	EE62			
EC01	3A	EE6A			
EC04	3C				
EC05	FE	40			
EC07	D4	EC11			
EC0A	E1				
EC0B	D1				
EC0C	C1				
EC0D	F1				
EC0E	FB				
EC0F	ED	4D			
EC11	01	1F99			
EC14	3A	EE45			
EC17	FE	4E			
EC19	C8				
EC1A	FE	52			
EC1C	28	11			
EC1E	FE	58			
EC20	C0				
EC21	3A	EE63			
EC24	B7				
EC25	C0				
EC26	3E	13			
EC28	32	EE63			
EC2B	CD	ED77			
EC2E	C9				
EC2F	01	1F99			
EC32	3E	05			
EC34	ED	79			
EC36	3A	EE54			
EC39	E6	FD			
EC3B	ED	79			
EC3D	C9				
EC3E	3A	EE45			
EC41	FE	4E			
EC43	C8				
EC44	FE	52			
EC46	28	12			
EC48	FE	58			
EC4A	C0				
EC4B	3A	EE63			
EC4E	B7				
EC4F	C8				
EC50	AF				
EC51	32	EE63			
EC54	3E	11			
EC56	CD	ED77			
EC59	C9				
EC5A	01	1F99			
EC5D	3E	05			
EC5F	ED	79			
EC61	3A	EE54			
EC64	F6	02			
EC66	ED	79			
EC68	C9				
EC69	CD	ECA4			
EC6C	47				
EC6D	3A	EE46			
EC70	FE	53			
EC72	20	24			
EC74	3A	EE61			
EC77	FE	7F			
EC79	20	1D			
			INTR2:	LD	HL,Q0 ;QUE !
				LD	(HEAD),HL
				LD	HL,QLEN
				INC	(HL)
				CP	XOFF
				JR	NZ,INTR3
				LD	(LASTX),A ;GET XOFF
				JR	INTR4
			INTR3:	CP	XON
				JR	NZ,INTR4
				XOR	A
				LD	(LASTX),A ;GET XON
			INTR4:	LD	A,(QLEN)
				INC	A
				CP	QMAX
				CALL	NC,SWAIT ;SEND WAIT
				POP	HL
				POP	DE
				POP	BC
				POP	AF
				EI	
				RETI	
			SWAIT:	LD	BC,ZSIO+1
				LD	A,(PARAM+0)
				CP	'N'
				RET	Z ;NOTHING
				CP	'R'
				JR	Z,SWAIT1
				CP	'X'
				RET	NZ
				LD	A,(XFLAG)
				OR	A
				RET	NZ
				LD	A,XOFF ;X CTRL
				LD	(XFLAG),A
				CALL	PPUTC0
				RET	
			SWAIT1:	LD	BC,ZSIO+1
				LD	A,5
				OUT	(C),A
				LD	A,(WR5+1)
				AND	11111101B ;RES RTS
				OUT	(C),A
				RET	
			SOK:	LD	A,(PARAM+0)
				CP	'N'
				RET	Z
				CP	'R'
				JR	Z,SOK1
				CP	'X'
				RET	NZ
				LD	A,(XFLAG)
				OR	A
				RET	Z
				XOR	A
				LD	(XFLAG),A
				LD	A,XON
				CALL	PPUTC0
				RET	
			SOK1:	LD	BC,ZSIO+1
				LD	A,5
				OUT	(C),A
				LD	A,(WR5+1)
				OR	00000010B ;SET RTS
				OUT	(C),A
				RET	
			XGETC:	CALL	PGETC ;GET CHAR
				LD	B,A ;SAVE
				LD	A,(PARAM+1)
				CP	'S'
				JR	NZ,XGETC4
				LD	A,(MASK)
				CP	1111111B
				JR	NZ,XGETC4



EC7B 3A EE65  
EC7E B7  
EC7F 78  
EC80 20 06  
EC82 FE 0E  
EC84 28 07  
EC86 18 10

EC88 FE 0F  
EC8A 20 06  
EC8C AF  
EC8D 32 EE65  
EC90 18 D7

EC92 FE 20  
EC94 38 02  
EC96 CB F8  
EC98 78  
EC99 C9

EC9A E5  
EC9B CD ECA4  
EC9E E1  
EC9F 77  
ECA0 23  
ECA1 36 00  
ECA3 C9

ECA4 CD 0330  
ECA7 CA EE2A  
ECAA 3A EE6A  
ECAD B7  
ECAE 28 F4

ECB0 F3  
ECB1 3A EE6A  
ECB4 5F  
ECB5 2A EE68  
ECB8 3A EE61  
ECBB A6  
ECBC 57  
ECBD 23  
ECBE E5  
ECBF 01 EEEB  
ECC2 B7  
ECC3 ED 42  
ECC5 E1  
ECC6 20 03  
ECC8 21 EE6B  
ECCB 22 EE68  
ECCE 7B  
ECCF 3D  
ECD0 32 EE6A  
ECD3 FE 30  
ECD5 D5  
ECD6 DC EC3E  
ECD9 FB  
ECDA D1  
ECDB 7A  
ECDC C9

ECDD EB  
ECDE E5  
ECDF 23  
ECE0 78  
ECE1 FE 02  
ECE3 DA EE26  
ECE6 05  
ECE7 0E 00

ECE9 E5  
ECEA C5  
EC EB CD 0330  
EC EE CA EE2A  
EC F1 3A EE6A  
EC F4 B7  
EC F5 28 F4  
EC F7 CD EBA0  
EC FA 28 0A

;S AND 7 BITS

LD A, (RKIN)  
OR A  
LD A, B  
JR NZ, XGETC1  
CP SO ;TO KANA MODE  
JR Z, XGETC2  
JR XGETC4

;XGETC1: CP SI  
JR NZ, XGETC3  
XOR A  
XGETC2: LD (RKIN), A ;CLEAR  
JR XGETC ;AGAIN

;XGETC3: CP 0A0H-080H  
JR C, XGETC4  
SET 7, B ;TO KANA  
XGETC4: LD A, B  
RET

;PGETC1: PUSH HL ;RETURN CHAR  
CALL PGETC  
POP HL  
LD (HL), A  
INC HL  
LD (HL), 0  
RET

;PGETC: CALL QBREAK  
JP Z, BREAK ;BREAK?  
LD A, (QLEN)  
OR A  
JR Z, PGETC

;DI  
LD A, (QLEN) ;RE-GET  
LD E, A ;SAVE QLEN  
LD HL, (TAIL)  
LD A, (MASK)  
AND (HL)  
LD D, A ;GET DATA

INC HL  
PUSH HL  
LD BC, Q1  
OR A  
SBC HL, BC  
POP HL  
JR NZ, GETC1  
LD HL, Q0 ;QUE !  
GETC1: LD (TAIL), HL  
LD A, E ;GET QLEN  
DEC A  
LD (QLEN), A  
CP QMAX-16  
PUSH DE  
CALL C, SOK ;SEND OK  
EI  
POP DE  
LD A, D  
RET

;LINPUT: EX DE, HL  
PUSH HL  
INC HL  
LD A, B  
CP 2  
JP C, ERR ;NO SPACE  
DEC B ;DEC COUNTER  
LD C, 0

;LIP1: PUSH HL  
PUSH BC ;SAVE COUNTER  
LIP1L: CALL QBREAK ;BREAK?  
JP Z, BREAK  
LD A, (QLEN)  
OR A  
JR Z, LIP1L  
CALL QEOF  
JR Z, LIP1L1

;GET EOF



ECFC	C1	POP	BC
ECFD	E1	POP	HL
ECFE	79	LD	A,C
ECFF	FE 0D	CP	CR
ED01	20 2A	JR	NZ,LIPL2
ED03	2B	DEC	HL
ED04	18 27	JR	LIPL2 ;SUTE CR
;			
ED06	CD EC69	LIP1L1: CALL	XGETC
ED09	C1	POP	BC
ED0A	E1	POP	HL
;1 LINE?			
ED0B	F5	PUSH	AF
ED0C	3A EE48	LD	A,(PARAM+3)
ED0F	FE 4C	CP	'L'
ED11	28 07	JR	Z,LIPL1
ED13	F1	POP	AF
ED14	FE 0D	CP	CR
ED16	28 15	JR	Z,LIPL2
ED18	18 0E	JR	LIP2
ED1A	79	LIPL1: LD	A,C
ED1B	FE 0D	CP	CR
ED1D	20 08	JR	NZ,LIPL3
ED1F	F1	POP	AF
ED20	FE 0A	CP	LF
ED22	20 04	JR	NZ,LIP2
ED24	2B	DEC	HL
ED25	18 06	JR	LIPL2
;			
ED27	F1	LIPL3: POP	AF
ED28	77	LIP2: LD	(HL),A
ED29	4E	LIP3: LD	C,(HL) ;LAST CHAR
ED2A	23	INC	HL
ED2B	10 BC	DJNZ	LIP1 ;FULL?
ED2D	D1	LIPL2: POP	DE
ED2E	B7	OR	A
ED2F	ED 52	SBC	HL,DE
ED31	EB	EX	DE,HL
ED32	1D	DEC	E
ED33	73	LD	(HL),E ;STORE LEN
ED34	C9	RET	
;			
;INPUT n CHAR			
ED35	D5	INPUTN: PUSH	DE ;SAVE ADDR
ED36	EB	EX	DE,HL
ED37	7E	LD	A,(HL)
ED38	23	INC	HL
ED39	B8	CP	B
ED3A	D2 EE26	JP	NC,ERR ;TOO SHORT
;A<B			
ED3D	05	DEC	B
ED3E	B7	OR	A
ED3F	28 12	JR	Z,IPN2 ;REQUEST 0 CHAR
ED41	47	LD	B,A ;COUNTER
ED42	0E 00	LD	C,0 ;LEN
;			
ED44	C5	IPN1: PUSH	BC
ED45	E5	PUSH	HL
ED46	CD EC69	CALL	XGETC
ED49	E1	POP	HL
ED4A	C1	POP	BC
ED4B	77	LD	(HL),A ;STORE CODE
ED4C	23	INC	HL
ED4D	0C	INC	C
ED4E	10 F4	DJNZ	IPN1
ED50	E1	POP	HL ;TOP ADDR.
ED51	71	LD	(HL),C ;STORE LEN
ED52	C9	RET	
;			
ED53	D1	IPN2: POP	DE
ED54	12	LD	(DE),A ;STORE 0
ED55	C9	RET	
;			
ED56	CD ED68	PRINT: CALL	PRINTS
ED59	3E 0D	LD	A,CR
ED5B	CD EDCD	CALL	PPUTC
ED5E	3A EE47	LD	A,(PARAM+2)
ED61	FE 4C	CP	'L'
ED63	C0	RET	NZ ;'C'
ED64	3E 0A	LD	A,LF ;'L'
ED66	18 65	JR	PPUTC



```

ED68 78
ED69 B7
ED6A C8
ED6B 1A
ED6C 13
ED6D C5
ED6E D5
ED6F CD ED93
ED72 D1
ED73 C1
ED74 10 F5
ED76 C9

ED77 F5
ED78 01 1F99
ED7B 3E 10
ED7D ED 79
ED7F ED 78
ED81 CB 57
ED83 28 F6

ED85 CD EE09
ED88 28 62
ED8A 3E 10
ED8C ED 79
ED8E 18 EB

ED90 7E
ED91 18 3A

ED93 47
ED94 3A EE46
ED97 FE 53
ED99 20 31

ED9B 3A EE61
ED9E FE 7F
EDA0 20 2A

EDA2 CB 78
EDA4 28 15
EDA6 3A EE64
EDA9 B7
EDAA 20 0A
EDAC C5
EDAD 3E 0F
EDAF CD EDCD
EDB2 32 EE64
EDB5 C1

EDB6 78
EDB7 E6 7F
EDB9 18 12

EDBB 3A EE64
EDBE B7
EDBF 28 0B

EDC1 C5
EDC2 3E 0E
EDC4 CD EDCD
EDC7 AF
EDC8 32 EE64
EDCB C1

EDCC 78
EDCD F5
EDCE 01 1F99
EDD1 CD 0330
EDD4 CA EE2A
EDD7 3E 10
EDD9 ED 79
EDDB ED 78
EDDD CB 57
EDDF 28 F0

EDE1 CD EE09
EDE4 28 06
EDE6 3E 10

```

```

;
PRINTS: LD A,B
        OR A ;CK LEN
        RET Z
PUTSL: LD A,(DE)
        INC DE
        PUSH BC
        PUSH DE
        CALL PUTC
        POP DE
        POP BC
        DJNZ PUTSL
        RET

;
PPUTC0: PUSH AF
        LD BC,ZSIO+1
PUTCL0: LD A,10H ;RESET STAT
        OUT (C),A
        IN A,(C) ;GET RR0
        BIT 2,A ;TX BUFF
        JR Z,PUTCL0

;
        CALL CHTXOK ;TX OK?
        JR Z,PUTOK
        LD A,10H ;STAT RESET
        OUT (C),A
        JR PUTCL0 ;LOOP

;
PPUTC1: LD A,(HL)
        JR PPUTC

;
PUTC: LD B,A
        LD A,(PARAM+1)
        CP 'S'
        JR NZ,PUTC0

;
        LD A,(MASK)
        CP 1111111B
        JR NZ,PUTC0

;
; 'S' AND 7 BITS
        BIT 7,B
        JR Z,NKANA ;NOT KANA
        LD A,(TKIN)
        OR A
        JR NZ,IKANA
        PUSH BC ;SAVE IT
        LD A,SI ;SHIFT IN
        CALL PPUTC
        LD (TKIN),A
        POP BC

;
IKANA: LD A,B
        AND 7FH ;SEND 7 BITS
        JR PPUTC

;
NKANA: LD A,(TKIN)
        OR A
        JR Z,PUTC0 ;NO KANA MODE

;
        PUSH BC ;SAVE IT
        LD A,SO ;SHIFT OUT
        CALL PPUTC ;PUT ANY HOW
        XOR A
        LD (TKIN),A
        POP BC ;B=IT

;
PUTC0: LD A,B ;DATA TO TX
PPUTC: PUSH AF
        LD BC,ZSIO+1
PUTCL: CALL QBREAK ;BREAK?
        JP Z,BREAK
        LD A,10H ;RESET STAT
        OUT (C),A
        IN A,(C) ;GET RR0
        BIT 2,A ;TX BUFF
        JR Z,PUTCL

;
        CALL CHTXOK ;TX OK?
        JR Z,PUTOK
        LD A,10H ;STAT RESET

```



EDE8	ED 79	OUT	(C),A	
EDEA	18 E5	JR	PUTCL	;LOOP
EDEC	0B	;		
EDED	D1	PUTOK:	DEC	BC ;DATA PORT
EDEE	3A EE61		POP	DE ;D=CHAR
EDF1	A2		LD	A,(MASK)
EDF2	ED 79		AND	D
			OUT	(C),A ;TX DATA
		;		
EDF4	3A EE6A		LD	A,(QLEN)
EDF7	B7		OR	A ;EMPTY?
EDF8	C8		RET	Z
		;		
		;	DROP XON,XOFF	
EDF9	2A EE68	PEND0:	LD	HL,(TAIL)
EDFC	7E		LD	A,(HL) ;NOZOKU
EDFD	FE 11		CP	XON
EDFF	28 03		JR	Z,PEND1
EE01	FE 13		CP	XOFF
EE03	C0		RET	NZ
EE04	CD EC9A	PEND1:	CALL	PGETC1 ;DROP
EE07	18 F0		JR	PEND0
		;		
EE09	57	CHTXOK:	LD	D,A ;SAVE RR0
EE0A	3A EE45		LD	A,(PARAM+0)
EE0D	FE 4E		CP	'N' ;NO CON
EE0F	C8		RET	Z
EE10	FE 52		CP	'R' ;RTS
EE12	20 05		JR	NZ,CHX
EE14	7A		LD	A,D
EE15	2F		CPL	;REVERSE
EE16	CB 6F		BIT	5,A ;RTS
EE18	C9		RET	
		;		
EE19	FE 58	CHX:	CP	'X'
EE1B	28 02		JR	Z,CHX1
EE1D	AF		XOR	A
EE1E	C9		RET	
EE1F	F3	CHX1:	DI	;ENGIMON
EE20	3A EE62		LD	A,(LASTX)
EE23	B7		OR	A
EE24	FB		EI	
EE25	C9		RET	
		;		
EE26	3E 05	ERR:	LD	A,5 ;ILL FUNC
EE28	DD E9		JP	(IX)
		;		
EE2A	F3	BREAK:	DI	
EE2B	C3 1FCA		JP	BBREAK
		;		
EE2E		CTR:	DS	1 ;CTC COUNTER
		;		
EE2F	58 52 4E 00	PLIST:	DB	'XRN',0
EE33	53 4E 00		DB	'SN',0
EE36	43 4C 00		DB	'CL',0
EE39	43 4C 00		DB	'CL',0
EE3C	4A 4E 00		DB	'JN',0
		;		
EE3F	4E 4E 4C 4C	PDFLT:	DB	'NNLLN',20H
EE43	4E 20			
		;		
EE45		PARAM:	DS	5 ;P AREA
EE4A		ECODE:	DS	1 ;5+1=6
		;		
EE4B		IOC:	DS	1 ;OPEN MODE FLAG
		;		
EE4C	18	SIOSA:	DB	00011000B ;WR0
EE4D	01 10	WR1:	DB	1,10H
EE4F	02 00	WR2:	DB	2,00H
EE51	04 00	WR4:	DB	4,00H
EE53	05 00	WR5:	DB	5,00H
EE55	06 00	WR6:	DB	6,00H
EE57	07 00	WR7:	DB	7,00H
EE59	03 00	WR3:	DB	3,00H
		;		
EE5B	18	SIOSB:	DB	18H
EE5C	01 00		DB	1,00H
EE5E	02 5C		DB	2,INTRSV
		;		
EE60	00	OPENF:	DB	0 ;OPEB FLAG
		;		



EE61	MASK:	DS	1	;DATA MASK
EE62	LASTX:	DS	1	;R FLAG
EE63	XFLAG:	DS	1	;T FLAG
	;			
EE64	TKIN:	DS	1	;R KANA FLAG
EE65	RKIN:	DS	1	;T KANA FLAG
	;			
EE66	HEAD:	DS	2	;QUE HEAD
EE68	TAIL:	DS	2	;QUE TAIL
EE6A	QLEN:	DS	1	;Q LENGTH
EE6B	Q0:	DS	QMAXX	
EEEB	Q1:			
	;			
	END			

リスト 7-2 通信プログラムのダンプリスト

EA00 C3 1E EA C3 84 EB C3 DD : 9D	EB70 EE 22 66 EE 22 68 EE 32 : 0E
EA08 EC C3 35 ED C3 68 ED C3 : AC	EB78 62 EE 32 63 EE 32 64 EE : 57
EA10 56 ED C3 98 EB C3 B9 EB : F0	-----
EA18 C3 9A EC C3 90 ED 3A 60 : 23	SUM: 07 6C 65 9E F3 F0 F7 30 1771
EA20 EE B7 28 07 D5 C5 CD 84 : BF	
EA28 EB C1 D1 3E FF 32 60 EE : 3A	EB80 32 65 EE C9 AF 32 60 EE : 7D
EA30 3D 20 FD F3 21 C6 EB 22 : 41	EB88 3A 4B EE FE 4F C0 3A 4A : 04
EA38 5C 00 FB EB 78 B7 CA 26 : 61	EB90 EE FE 20 C8 CD CD ED C9 : 24
EA40 EE 05 7E 23 FE 49 28 0B : 0E	EB98 3A 6A EE 77 23 36 00 C9 : 2B
EA48 FE 4F 28 07 FE 43 28 03 : E8	EBA0 3A 6A EE B7 C8 3A 4A EE : 83
EA50 C3 26 EE 32 4B EE 78 D6 : 90	EBA8 FE 20 C8 E5 2A 68 EE 46 : 91
EA58 04 DA 26 EE FE 07 D2 26 : EF	EBB0 E1 B8 28 02 AF C9 F6 FF : 30
EA60 EE 4F 7E 23 D6 30 FE 07 : E9	EBB8 C9 CD A0 EB 11 00 00 28 : 5A
EA68 D2 26 EE 16 D0 D6 02 F5 : 99	EBC0 01 1B 73 23 72 C9 F3 F5 : D5
EA70 30 01 AF B7 28 05 CB 3A : C9	EBC8 C5 D5 E5 01 99 1F 3E 01 : 77
EA78 3D 18 F8 7A 32 2E EE F1 : 06	EBD0 ED 79 ED 78 0B ED 78 2A : 65
-----	EBD8 66 EE 77 23 E5 01 EB EE : AD
SUM: 1A E2 8C E2 74 31 D8 D6 6C5A	EBE0 B7 ED 42 E1 20 03 21 6B : 76
	EBE8 EE 22 66 EE 21 6A EE 34 : 11
EA80 16 40 30 07 16 80 3C 28 : 87	EBF0 FE 13 20 05 32 62 EE 18 : D0
EA88 02 16 C0 5A 7E 23 16 00 : E9	EBF8 08 FE 11 20 04 AF 32 62 : 7E
EA90 FE 4E 28 0B 14 FE 4F 28 : 08	-----
EA98 06 14 FE 45 C2 26 EE 7B : AE	SUM: 3A 9E FD 42 12 B4 78 4C 99CD
EEA0 B2 5F 7E 23 D6 35 DA 26 : BD	
EEA8 EE FE 04 D2 26 EE F5 16 : E1	EC00 EE 3A 6A EE 3C FE 40 D4 : CE
EAB0 E0 B7 28 05 CB 22 3D 20 : 0E	EC08 11 EC E1 D1 C1 F1 FB ED : 49
EAB8 FB 7A 2F 32 61 EE F1 B7 : CD	EC10 4D 01 99 1F 3A 45 EE FE : 71
EAC0 EA C5 EA EE 03 0F 0F F6 : 9E	EC18 4E C8 FE 52 28 11 FE 58 : F5
EAC8 01 32 5A EE 0F F6 80 F6 : F6	EC20 C0 3A 63 EE B7 C0 3E 13 : 13
EAD0 02 F6 08 32 54 EE 7E 23 : 15	EC28 32 63 EE CD 77 ED C9 01 : 7E
EAD8 D6 31 DA 26 EE FE 03 D2 : C8	EC30 99 1F 3E 05 ED 79 3A 54 : EF
EAE0 26 EE 3C 87 87 B3 32 52 : 95	EC38 EE E6 FD ED 79 C9 3A 45 : 7F
EAE8 EE C5 E5 21 3F EE 11 45 : 3C	EC40 EE FE 4E C8 FE 52 28 12 : 8C
EAFO EE 01 06 00 ED B0 EB D9 : 56	EC48 FE 58 C0 3A 63 EE B7 C8 : 20
EAFF E1 C1 11 2F EE AF 08 79 : 00	EC50 AF 32 63 EE 3E 11 CD 77 : C5
-----	EC58 ED C9 01 99 1F 3E 05 ED : 9F
SUM: 3D D9 4D E8 87 EB D2 A8 CC3B	EC60 79 3A 54 EE F6 02 ED 79 : 53
	EC68 C9 CD A4 EC 47 3A 46 EE : DB
EB00 B7 28 1F FE 06 20 02 0D : 31	EC70 FE 53 20 24 3A 61 EE FE : 1C
EB08 08 46 23 1A 13 B7 CA 26 : 45	EC78 7F 20 1D 3A 65 EE B7 78 : 78
EB10 EE B8 20 F7 D9 12 D9 1A : 9B	-----
EB18 13 B7 20 FB D9 13 D9 0D : B7	SUM: 5A 5C 15 9E 8D 4E 2B DF 9564
EB20 20 E7 08 B7 28 10 7E FE : 7A	
EB28 40 DA 26 EE FE 80 D2 26 : A4	EC80 20 06 FE 0E 28 07 18 10 : 89
EB30 EE E6 1F D9 12 D9 F3 CD : 77	EC88 FE 0F 20 06 AF 32 65 EE : 67
EB38 6A EB 01 A9 1F 3E 47 ED : 90	EC90 18 D7 FE 20 38 02 CB F8 : 0A
EB40 79 3A 2E EE ED 79 21 4C : A2	EC98 78 C9 E5 CD A4 EC E1 77 : DB
EB48 EE 01 99 1F 3E 0F CD 63 : 24	ECA0 23 36 00 C9 CD 30 03 CA : EC
EB50 EB 21 5B EE 01 9B 1F 3E : 4E	ECA8 2A EE 3A 6A EE B7 28 F4 : 7D
EB58 05 CD 63 EB 3E FF 32 60 : EF	ECB0 F3 3A 6A EE 5F 2A 68 EE : 64
EB60 EE FB C9 04 ED A3 3D 20 : A3	ECB8 3A 61 EE A6 57 23 E5 01 : 8F
EB68 FA C9 AF 32 6A EE 21 6B : 88	ECC0 EB EE B7 ED 42 E1 20 03 : C3



ECC8	21	6B	EE	22	68	EE	7B	3D	:	AA	ED98	53	20	31	3A	61	EE	FE	7F	:	AA
ECD0	32	6A	EE	FE	30	D5	DC	3E	:	A7	EDA0	20	2A	CB	78	28	15	3A	64	:	68
ECD8	EC	FB	D1	7A	C9	EB	E5	23	:	EE	EDA8	EE	B7	20	0A	C5	3E	0F	CD	:	AE
ECE0	78	FE	02	DA	26	EE	05	0E	:	79	EDB0	CD	ED	32	64	EE	C1	78	E6	:	5D
ECE8	00	E5	C5	CD	30	03	CA	2A	:	9E	EDB8	7F	18	12	3A	64	EE	B7	28	:	14
ECF0	EE	3A	6A	EE	B7	28	F4	CD	:	20	EDC0	0B	C5	3E	0E	CD	CD	ED	AF	:	52
ECF8	A0	EB	28	0A	C1	E1	79	FE	:	D6	EDC8	32	64	EE	C1	78	F5	01	99	:	4C
-----											EDD0	1F	CD	30	03	CA	2A	EE	3E	:	3F
SUM:	58	3A	50	EE	95	E4	39	BE	CDF1		EDD8	10	ED	79	ED	78	CB	57	28	:	25
-----											EDE0	F0	CD	09	EE	28	06	3E	10	:	30
ED00	0D	20	2A	2B	18	27	CD	69	:	F7	EDE8	ED	79	18	E5	0B	D1	3A	61	:	DA
ED08	EC	C1	E1	F5	3A	48	EE	FE	:	F1	EDF0	EE	A2	ED	79	3A	6A	EE	B7	:	3F
ED10	4C	28	07	F1	FE	0D	28	15	:	B4	EDF8	C8	2A	68	EE	7E	FE	11	28	:	FD
ED18	18	0E	79	FE	0D	20	08	F1	:	C3	-----										
ED20	FE	0A	20	04	2B	18	06	F1	:	66	SUM:	CA	40	7A	D2	2F	72	2F	93	6DF0	
ED28	77	4E	23	10	BC	D1	B7	ED	:	29	-----										
ED30	52	EB	1D	73	C9	D5	EB	7E	:	D4	EE00	03	FE	13	C0	CD	9A	EC	18	:	3F
ED38	23	B8	D2	26	EE	05	B7	28	:	A5	EE08	F0	57	3A	45	EE	FE	4E	C8	:	C8
ED40	12	47	0E	00	C5	E5	CD	69	:	47	EE10	FE	52	20	05	7A	2F	CB	6F	:	58
ED48	EC	E1	C1	77	23	0C	10	F4	:	38	EE18	C9	FE	58	28	02	AF	C9	F3	:	B4
ED50	E1	71	C9	D1	12	C9	CD	68	:	FC	EE20	3A	62	EE	B7	FB	C9	3E	05	:	48
ED58	ED	3E	0D	CD	CD	ED	3A	47	:	40	EE28	DD	E9	F3	C3	CA	1F	15	58	:	D2
ED60	EE	FE	4C	C0	3E	0A	18	65	:	BD	EE30	52	4E	00	53	4E	00	43	4C	:	D0
ED68	78	B7	C8	1A	13	C5	D5	CD	:	8B	EE38	00	43	4C	00	4A	4E	00	4E	:	75
ED70	93	ED	D1	C1	10	F5	C9	F5	:	D5	EE40	4E	4C	4C	4E	20	CD	83	21	:	C5
ED78	01	99	1F	3E	10	ED	79	ED	:	5A	EE48	47	C3	B3	20	18	01	10	02	:	08
-----											EE50	00	04	00	05	00	06	00	07	:	16
SUM:	0D	24	66	AA	33	B7	5D	11	8D3D		EE58	00	03	00	18	01	00	02	5C	:	7A
-----											EE60	00								:	00
ED80	78	CB	57	28	F6	CD	09	EE	:	7C	-----										
ED88	28	62	3E	10	ED	79	18	EB	:	41	SUM:	B8	97	F1	8A	CD	80	F9	BF	E0FD	
ED90	7E	18	3A	47	3A	46	EE	FE	:	83	-----										

リスト 7-3 サンプルプログラム

```

100 CLEAR &HEA00
110 IF MEM$(&HEA00,3)<>HEXCHR$("C3 1E EA") LOADM "RS.OBJ"
120 DEFUSR0=&HEA00          : 'OPEN   :STR
130 DEFUSR1=&HEA00+3        : 'CLOSE  : *
140 DEFUSR2=&HEA00+6        : 'LINPUT :STR
150 DEFUSR3=&HEA00+9        : 'INPUTN :STR
160 DEFUSR4=&HEA00+12       : 'PRINTS :STR
170 DEFUSR5=&HEA00+15       : 'PRINT  :STR
180 DEFUSR6=&HEA00+18       : 'LOC    :INT
190 DEFUSR7=&HEA00+21       : 'EOF    :INT
200 DEFUSR8=&HEA00+24       : 'PGETC1 :INT
210 DEFUSR9=&HEA00+27       : 'PPUTC1 :INT
220 '
230 A$=USR0("O6N83XNCCNZ") : 'OPEN
240 A$=USR1("")              : 'CLOSE
250 C=10:A$=USR2(STRING$(C,".")):'LINPUT
260 C=10:A$=USR3(CHR$(C)+STRING$(C,".")):'INPUTN
270 A$=USR4("STRINGS")      : 'SEND STRING
280 A$=USR5("ONE LINE")     : 'SEND 1 LINE
290 A=USR6(0)                : 'GET BUF SIZE
300 A=USR7(0)                : 'EOF ?
310 A=USR8(0)                : 'GET 1 BYTE
320 A=USR9(0)                : 'PUT 1 BYTE

```

リスト 7-4 簡単なターミナルプログラム(100～210行はリスト 7-3と同じ)

```

100 CLEAR &HEA00
110 IF MEM$(&HEA00,3)<>HEXCHR$("C3 1E EA") THEN LOADM "RS.OBJ"
120 DEFUSR0=&HEA00          : 'OPEN   :STR
130 DEFUSR1=&HEA00+3        : 'CLOSE  : *
140 DEFUSR2=&HEA00+6        : 'LINPUT :STR

```



```

150 DEFUSR3=&HEA00+9      : 'INPUTN :STR
160 DEFUSR4=&HEA00+12     : 'PRINTS :STR
170 DEFUSR5=&HEA00+15     : 'PRINT  :STR
180 DEFUSR6=&HEA00+18     : 'LOC    :INT
190 DEFUSR7=&HEA00+21     : 'EOF    :INT
200 DEFUSR8=&HEA00+24     : 'PGETC1 :INT
210 DEFUSR9=&HEA00+27     : 'PPUTC1 :INT
220 '100 - 210 キョウハリスト3 ノモノヲツカウノデアル
230 A$=USR0("C6N83XNCCNZ") : 'OPEN
240 L$=""
250 IF USR7(0) THEN PRINT "GET END":END 'EOF
260 A=USR6(0):IF A=0 THEN 300 'LOC > 0?
270 A$=USR3(CHR$(1,0))
280 A$=RIGHT$(A$,1):A=ASC(A$) 'GET AND DISPLAY
290 IF (A>=&H20) OR (A=13) THEN PRINT A$;:GOTO 260
300 C$=INKEY$(0) 'READ KEY
310 IF (C$=CHR$(0)) OR (L$=C$) THEN 340 'YUUKOU ?
320 IF C$=CHR$(13) THEN A$=USR5(""):GOTO 340
330 A$=USR4(C$) 'TX IT
340 L$=C$:GOTO 250

```

ここでこの章のプログラムの手抜きなどについて書いておくことにする。

まずは XON, XOFF 制御についてである。やっててだんだん面倒になってきたので、「送られてきた XON/XOFF はバッファにたまる」ようになっている。ただしバッファのトップが XON もしくは XOFF であったならば、「こちらからデータを送るときにチェックして捨てる」ようにしている。もしも向こうから「XON/XOFF と他のコードも混ぜて送ってきたら」受信に混ざりこむことになる。必要ならば表示ルーチンなどで取り除いていただきたい。これは、バイナリデータとして、たまたま XON/XOFF (11<sub>H</sub>/13<sub>H</sub>) が送られてくる場合もあるのでこうしたのだが、本当はどうすべきだったのでしょうか。また CLOSE 時の動作もよく分かんないのである。turbo BASIC では「O」でオープンしたときだけエンドコードを送るようになっているようであるが、「C」では送らないようである。またその逆のエンドコードを受け取ったときの動作も不明瞭である。turbo BASIC では、LINPUT がエンドコードを受け取ったときは 1 行が終了したとみなしてリターン、INPUT\$ で入力文字数を指定した場合はエンドコードが来たとしても、指定文字数までリターンしない、などとなっている。こころへんの細かな動作は、BASIC のマニュアルにきっちり書いてあるわけではないので、なんだかよくわかんないのである。

おっとここで思い出した。この章のプログラムでは、「EOF はこれから読み出そうとしているデータがエンドコードだったら -1 を返す」ようになっている。よって不幸にして INPUT\$ などその「エンドコードを読み出してしまったりすると」EOF でなくなってしまうのである。注意していただきたい。しかし LINPUT でエンドコードは読み出さないようになっているから、エンドコードが一度来たら、それ以降はずっと EOF である。

なんだかんだ書いたが、はっきり言って結局は USR8, USR9 の「生データ読み出し、書き込み」が最終兵器となるような気がする。つらつら考えるに、この二つの機能+LOC 関数+OPEN コマンド+あぶったイカさえあれば、ほかには何もいらないうえな気がしてしまうのである。というわけで、本当にやるのならば大いに改造していただきたいと思うしだいである。



第

8

章

DMA



DMAはヘビー級である



## ■第8章

# DMAはヘビー級である……………

この章では、Z80 ファミリーの中でも特に「荒技使い」との呼び声が高い Z80 DMA についてやるのである。残念ながらこの石は turbo にしか付いていないが、それはそれで仕方がないのである。ちなみにこの石をノン turbo の X1 に付けようとする場合は、本体基板に対する改造が必要となる。CTC や SIO のようにボードに載せて I/O スロットにさすということはできないのである。

DMA というのは「Direct Memory Access」の頭文字を取って作られた言葉である。その点からすれば、DMA 動作をする LSI は DMAC (DMA Controller) と呼ばれるべきなのだが、Zilog 社はアルファベット 3 文字にしたかったようである。それはさておき、この石はデータ転送を主要な業務とする LSI である。M は Memory の M であるが、これは別にメインメモリとは限らず、I/O でもよいのである。この点、誤解のないように。

さて、Z80 ファミリーには PIO, CTC, SIO などがあるが、DMA はこの中でも CPU と互角の地位にあり、他の周辺 LSI とは一線を画しているのである。もしも DMA に聞いてみたならば、「わしゃー、偉いんでい！」と答えるはずである。それはなぜかというと、DMA は「データ転送に関しては CPU と同等の立場に立っている」からなのである。うーん、これは非常に曖昧な表現である。曖昧ついでに書いてしまうが、DMA がどれだけ CPU と同等かということ、「DMA が動いている間は、CPU が止まらざるを得ない」ほどなのである。正確に言うと、CPU はバス(アドレスバス、データバス、コントロールバスの三種の神器)を DMA に明け渡してしまうのである。しかし、ここではハードウェアがどーのこーのという話はあまりしないことにする。詳しく勉強したい人は参考文献 4 である。

まず何よりも DMA が何をするかについて説明しておく。

DMA にできることといえば、後にも先にも、データを CPU より高速に転送/サーチできるということ以外にはない(CPU と同じか、より遅いのならば LDIR なり OUTI なりやっつけてしまえばよい)。この「速い」ということは非常に大事なことなのである。これが一番はっきりと表面に出てくるのは、8 インチドライブや、5 インチの 2HD を使おうとしたときである。どういうことかということ、これらのディスクのデータ転送速度はやたらと速くて(2D, 2DD の約 2 倍)、4MHz の Z80 では追い付かないのである。具体的には、たとえば読む場合である。X1 では、ディスクからデータを読むということは、FDC(フロッピーディスクコントローラ：第 9 章を参照)に適切なコマンドを与えた後に、FDC のステータスを見張っていて、FDC が「データを受け取ってくれ！」と言ってきたら、データを IN 命令で持ってくるということなのであるが、5 インチの倍密度(2D や 2DD がそう)の場合なら Z80 でもゆうゆう処理できるのだが、8 インチ、2HD などになると、お手上げになってしまうのである。人から聞いた話によると 8 インチ FD の場合「Z80 の 4MHz ではギリギリ」なのだそう。私は実際に、DMA を持っていない MZ-80B(もちろん CPU は 4MHz



の Z80) に 8 インチドライブをつなごうとして失敗した人を知っている。そういうわけで、DMA が X1 turbo に付いているのである。

以上のことから分かるように、DMA とフロッピーディスクとは深い関係にある。しかし DMA の偉いところは、決して「フロッピーディスクのリード/ライト専用」になっていないことである。つまり、単純なデータ転送や、データサーチなら、フロッピーディスクに関係なくとも高速に行なわせることができるのである。実にオイシイ石なのである。

なお、turbo など で 2HD ディスクを使っている場合は FD の動作がおかしくなることもあり得るので、以下のサンプルプログラムを使う間は、ディスクは 2D モードにしておくことをお勧めする。

## DMAの基礎

DMA の基本動作には次の六つがある。

### ① メモリ→メモリ転送

これは結局のところブロック転送である。LDIR, LDDR と基本的には同じだが、

- 1) 一定以上のバイト数を転送するなら DMA の方が速い。
- 2) ソースアドレスをインクリメントし、ディスティネーションアドレスをデクリメントする「順序逆転」などの変態行為を行なわせることができる。

……の点において味がある。

### ② メモリ→I/O 転送

つまり、グラフィック VRAM の書き込み、読み出しである。もちろん、テキスト VRAM でもよい。矩形領域（画面上の四角）の中を読み出すなどの高等な機能は当然ないが、知恵を絞れば泉が湧くのである。

### ③ I/O→I/O 転送

最初に「スクロール」という言葉がひらめいたのなら、貴兄はその筋である。そのほかにも、青画面→赤画面転送など、一体何に使ったらよいのか悩んでしまう用途も考えられる。大容量 RAM ボード (EMM) も I/O 空間につながっているので、その方向でも使う価値がある。

### ④ メモリサーチ

大きなデータ中のサーチは結構時間がかかってしまうものであるが、かといって DMA を使うのは大袈裟な気もする。サーチするデータは 1 バイトだけであるがマスクをかけることもできる。たとえば、

&H 1?0??1?? ( ? は無視する )

というパターンをサーチできる。特殊な形式のデータを扱う際には役に立つかも。

### ⑤ I/O サーチ

基本的には、やはりグラフィック VRAM のサーチであろうか？ しかし「0 でないデータを捜せ」などではできないのでいまいちのようである。となればテキスト VRAM のサーチということになるが、たかだか 2K バイトをサーチするために DMA を使うのは気がひけ



る。こうなったら EMM の中でもサーチするしかない。

## ⑥ 転送とサーチの合わせ技

①～②の転送とサーチを組み合わせることが可能なのである。つまり、転送しながらあるデータを見つけたら止まれ、というようなことができる。う～ん何に使おう。

以上が基礎教養である。私はとんでもないことに、この六つの基本動作すべてについてサンプルプログラムを書こうとしているのである。DMA という石がどれだけポップでコープンするものなのかを示したいともくろんでいるのである。

# DMAは爆発である

では、DMA を使うということが、どんな雰囲気であるのかを簡単に示しておく。

リスト 8-1 は普通の機械語プログラムで、画面をすべて一つの文字で埋めるプログラムである。やっぱり、速すぎるので 256 回実行している。

リスト 8-1 DMA を使わずに

			.Z80	
			.PHASE	0E00H
07D0		;	SIZE	EQU 80*25
0E00	26 00	;	START:	LD H,0 ;counter1
0E02	11 07D0	;	LOOP1:	LD DE,SIZE ;counter2
0E05	01 3000			LD BC,3000H
0E08	ED 61	;	LOOP2:	OUT (C),H
0E0A	03			INC BC
0E0B	1B			DEC DE
0E0C	7A			LD A,D
0E0D	B3			OR E ;DE=0 ?
0E0E	C2 0E08			JP NZ,LOOP2
0E11	24	;		INC H
0E12	C2 0E02			JP NZ,LOOP1
0E15	C9	;		RET
		;		END

それに対して、リスト 8-2 は DMA を使ってみたものである。分かりやすいように特別なテクニックは使っていない。これも同じく 256 回実行している。本当は DMA では似たことを続けて繰り返す場合にはもう少し速くなるのだが、そこまではやっていない。

リスト 8-2 DMA してみたプログラム

			.Z80	
			.PHASE	0E020H
07D0		;	SIZE	EQU 80*25
E020	3E 00	;		LD A,0 ;XOR A...COUNTER
E022	F5	;	LOOP3:	PUSH AF
E023	32 E05F			LD (DATAAD),A
E026	CD E035	;	MEIOSI:	CALL RESDMA ;RESET DMA
E029	21 E050			LD HL,DMAD ;LOAD TOP OF COMMANDS
E02C	CD E043			CALL WDMA ;SEND COMMANDS



```

E02F    F1                ; POP    AF        ;POP COUNTER
E030    3C                INC    A          ;0 TO 255
E031    C2 E022           JP     NZ,LOOP3
E034    C9                RET
;
E035    21 E03C           RESDMA: LD    HL,RESDAT
E038    CD E043           CALL   WDMA
E03B    C9                RET
;
E03C    06                RESDAT: DEFB    6          ;NUMBER OF COMMAND
E03D    C3 C3 C3          DEFB    0C3H,0C3H,0C3H
E040    C3 C3 C3          DEFB    0C3H,0C3H,0C3H
;
;WRITE DMA
;HL=TOP OF COMMANDS
;
E043    56                WDMA:  LD    D,(HL)
E044    01 1F80           LD     BC,1F80H          ;DMA PORT
E047    23                INC    HL
E048    04                REGWR: INC    B
E049    ED A3            OUTI
E04B    15                DEC    D
E04C    C2 E048           JP     NZ,REGWR
E04F    C9                RET
;
E050    0E                DMAD:  DEFB    0EH          ;NUMBER OF COM.
E051    7D                DEFB    7DH          ;(D7=0)AND(D1D0<>00)=WR0
E052    E05F             S_TOP:  DEFW    DATAAD    ;SUB0,1=PORT A TOP ADDR.
E054    07CF             LEN:    DEFW    SIZE-1      ;LENGTH
E056    24                S_KIND: DEFB    24H
;00100100B             :0??? ?100 = WR1 (PORT A MODE)
E057    18                D_KIND: DEFB    18H
;00011000B             :0??? ?000 = WR2 (PORT B MODE)
;                        SKIP WR3
E058    CD                DEFB    0CDH
;11001101B             :1?????01 = WR4 (WR0 FOR PORT B)
E059    3000             D_TOP:  DEFW    3000H        ;PORT B TOP ADDR
E05B    9A                DEFB    9AH          ;WR5 READY=HIGH
E05C    CF                DEFB    0CFH        ;WR6 (LOAD)
E05D    B3                DEFB    0B3H        ;WR6 (FORCE READY)LOAD)
E05E    87                DEFB    087H        ;WR6 (ENABLE DMA)
;
E05F    00                DATAAD: DEFB    0          ;DUMMY
;
END

```

実行して試してみるにはリスト 8-3 である。最初に普通の機械語で組んだものを実行する。何かキーを押せば、BEEP 音がして画面がくるくると書き換わる。CLS の直後は全画面のアトリビュートが縦 2 倍モードになっていることに注意。次にキーを押せば、DMA 版が走る。明らかに速いことが分かるだろう。最後に同じプログラムを BASIC で組んだものも入れておいた。すばらしく遅いことは言うまでもない。

リスト 8-3 普通の機械語と DMA を速度比較

```

100 CLS4
110 DEFINT A-Z
120 CLEAR &HDEEE
130 MEM$(&HE000,16)=HEXCHR$("26 00 11 D0 07 01 00 30 ED 61 03 1B 7A B3 C2 08")
140 MEM$(&HE010, 6)=HEXCHR$("E0 24 C2 02 E0 C9")
150 MEM$(&HE020,16)=HEXCHR$("3E 00 F5 32 5F E0 CD 35 E0 21 50 E0 CD 43 E0 F1")
160 MEM$(&HE030,16)=HEXCHR$("3C C2 22 E0 C9 21 3C E0 CD 43 E0 C9 06 C3 C3 C3")
170 MEM$(&HE040,16)=HEXCHR$("C3 C3 C3 56 01 80 1F 23 04 ED A3 15 C2 48 E0 C9")
180 MEM$(&HE050,16)=HEXCHR$("0E 7D 5F E0 D0 07 24 18 CD 00 30 9A CF B3 87 00")
190 PRINT"NORMAL":GOSUB"SOMEKEY":BEEP:CALL &HE000:BEEP
200 PRINT"DMA" :GOSUB"SOMEKEY":BEEP:CALL &HE020:BEEP
210 PRINT"BASIC" :GOSUB"SOMEKEY":BEEP
220 FOR I=0TO255
230 FORA=&H3000TO&H3000+25*80-1:OUTA,I:NEXT
240 NEXT:BEEP
250 END

```



```

260 LABEL "SOMEKEY"
270 IF INKEY$="" GOTO 270
280 RETURN

```

リスト 8-2 を見ると分かるだろうが、DMA を動かすには 1F8\*<sub>H</sub> 番地にコマンドを OUT してやればよい。逆に DMA からデータを受け取ることも可能で、そのときは IN 命令を使う。アドレス中の \* は何でもかまわないが、プログラム中では気分よく 1F80<sub>H</sub> になっている。

ところで、それじゃ DMA に与えることのできるコマンドはたった 1 バイトだけなのか？ という疑問が起こるはずである。もちろんそんなことはない。コマンドの種類は、大きく分けて 7 種類あり、それぞれのコマンドは「1 バイト中のこのビットが立っていたら、この種類」というふうに決まっているのである。簡単に言うなら、CPU と DMA の間であらかじめ決まったフォーマットで 8 ビットパラレルの通信を行なうようなものである。

## DMA は賢いのである

ではそういう意味も含めて、リスト 8-4 である。これはオール BASIC で書いてある。要は、DMA にコマンドを送ってやればよいのであるから、別に機械語を使わなくてもよいのである。唯一の問題が DMA を使う最大の利点＝スピードがなくなってしまうことである。ここではサンプルだからこれでよいのだ。

リスト 8-4 オール BASIC で DMA を使ってみる

```

100 CLEAR&HFFFF
110 POKE&HE000,ASC("X"): '画面を埋める文字
120 RESET$=HEXCHR$("C3 C3 C3 C3 C3 C3"): 'リセット データ
130 DMA$=RESET$+HEXCHR$("7D 00 E0 CF 07 24 18 CD 00 30 9A CF B3 87")
140 GOSUB "SETDMA"
150 END
160 '
170 LABEL "SETDMA"
180   FOR I=1 TO LEN(DMA$): OUT&H1F80,ASC(MID$(DMA$,I,1)):NEXT
190 RETURN

```

ではプログラムの説明に移る。このプログラムは、全画面に「X」を表示するものである。まず最初に CLEAR 命令で領域を確保している。プログラムでは &HE000 から &HEFFF までの 4K バイトを確保しているが、本当に必要なのはたった 1 バイトなのであった。なぜ 1 バイトが必要かというと、このプログラムでやっているメモリ→I/O 転送ではソース（転送元）をアドレスで指定する必要があるためなのだ。プログラムでは、そのソースは &HE000 に割り当ててあり、そこに ASC("X") を POKE している。

次が RESET\$=～となって、&HC3 が 6 個並んでいる。これは DMA に送るリセットコマンドである。DMA を初期化するにはとにかく 6 個のリセットコマンドを送ってやれば、DMA がどんな状態になっていても成功する。これは最悪の場合であって、本当なら 6 個も送る必要はない。しかし安全のために 6 個送っている。



次に  $DMA\$ = RESET\$ + \sim$  となっている部分が、主役のコマンドである。その後、`GOSUB "SETDMA"` で、 $DMA\$$  に入っている  $\&HC3 \sim \&H87$  までの 20 個の命令を DMA に送っている。

そこで肩慣らしにリスト 8-4 に出ている DMA のコマンドの解説を行なう。

### ① 概要

DMA へのコマンドは、大きく分けて 7 種類ある。それぞれ、WR0～WR6 と呼ばれている。WR とはライトレジスタ (Write Register) という意味で、早い話、DMA へコマンドを送るというのは、この七つのレジスタに値を書き込むこと、と解釈すればよい。なお、それぞれの WR の下にはサブレジスタというものがあり、多少複雑になっているが、別に難しいものではない。それぞれのコマンドの分類の仕方だけを図 8-1 に示す。全部で 7 種類なのだから左端（もしくは右端）の 3 ビットで判別できるようにすればいいようなものだが、きっとそこには Zilog などの複雑な事情というものが存在するのであろう。さらに細かな意味や内容を知りたい場合は図 8-2 ～ 8-9 を見ていただきたい。

### ② $C3_H = \&H11000011$

図 8-1 と比べれば、WR6 であることが分かるだろう。意味は前述のように RESET である。

### ③ $7D_H = \&B01111101$

MSB（左端）が 0 で、右端の二つが 00 でないから WR0 である。そこで図 8-2 を見る。D<sub>1</sub>、D<sub>0</sub> はデータの処理形式を指定する。ここでは「01」だからトランスファー（転送）である。D<sub>2</sub> (= 1) はデータの転送方向である。DMA は二つのポート（A と B）を持っ

図 8-1 DMA ライトレジスタ分類早見表

	D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>
WR0	0	?	?	?	?	?		
							00でない←	
WR1	0	?	?	?	?	1	0	0
WR2	0	?	?	?	?	0	0	0
WR3	1	?	?	?	?	?	0	0
WR4	1			?	?	?	0	1
				→11はダメ				
WR5	1	0	?	?	?	0	1	0
WR6	1	?	?	?	?	?	1	1

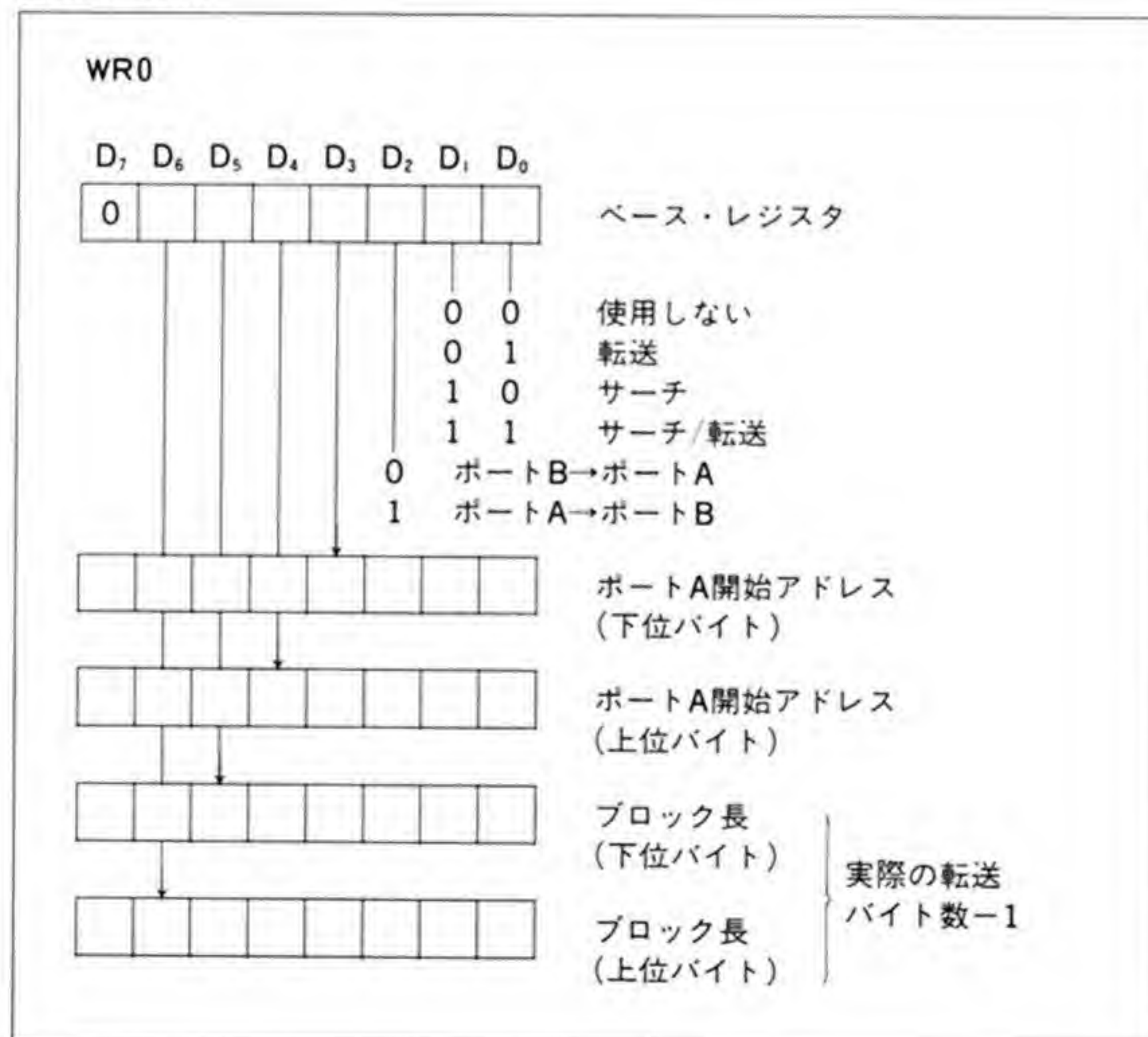


ていて、ソース（転送元）とディスティネーション（転送先）に割り当てるのである。A と B の間には、別にどうという違いはないので、迷う必要はない。ここでは A → B の方向にしている。D<sub>6</sub>, D<sub>5</sub>, D<sub>4</sub>, D<sub>3</sub>はポート A の開始アドレスと、ブロック長（処理バイト数 - 1）を設定するかどうかのフラグになっている（Z80 DMA の用語ではポインタビットという）。1であれば新しく設定するということになる。0の場合は、設定しないということであり、その場合は以前に設定されたものが使われることになる。図 8-2 から分かるように、それぞれを上位、下位に分けて設定するようになっていて、新しく設定するとなれば、LSB（右の方）から指定した順に 1F80<sub>H</sub>に OUT してやればよい。ここでは 4 ビットとも 1 であるから、全部指定することになる。つまり、00<sub>H</sub>, E0<sub>H</sub>, CF<sub>H</sub>, 07<sub>H</sub>が、それぞれサブレジスタの 0 ~ 3 に設定される。処理バイト数が 07CF<sub>H</sub> = 80 × 25 - 1 であることに注意。1 引いておくのだ。念のために言うが、もしブロック長の下位を設定しない場合は、2D<sub>H</sub> = &HB01011101 の後に、00<sub>H</sub>, E0<sub>H</sub>, 07<sub>H</sub>とすればよい。

#### ④ 24<sub>H</sub> = &B00100100

図 8-1 より WR1 である。D<sub>3</sub>により、ポート A をメモリにするか、I/O にするかを決定する（ポート B は WR2 で決める）。D<sub>5</sub>, D<sub>4</sub>は 1 バイト転送した後、ポート A のアドレスをどう変えるかを指定するものである。図 8-3 にあるように、- 1, + 1, 固定（変化させない）の三つの方式を指定できる。ここでは、テキスト VRAM を 1 種類の文字で埋めるのであるから、ソースであるポート A のアドレスは E000<sub>H</sub>に固定することになる。10, 11 どちらでもよいが、縁起物だから 10 にしてある。次に D<sub>6</sub>であるが、これはポート A へのア

図 8-2 WR0





クセスのタイミングを変える場合に1にする。その場合、次に1バイトを送ってやればサブレジスタに値を設定できる。ところがどっこい、turboではこの部分は使えないのである。使えたなら、さらに転送速度を上げることができるのであるが仕方がない。

#### ⑤ $18_H = \&B00011000$

WR2である。これは指定の対象がポートBになった点を除いてWR1と同じ使い方である。テキストVRAMへ書き込むのだから、ポートBはI/Oで、インクリメントにしてある。

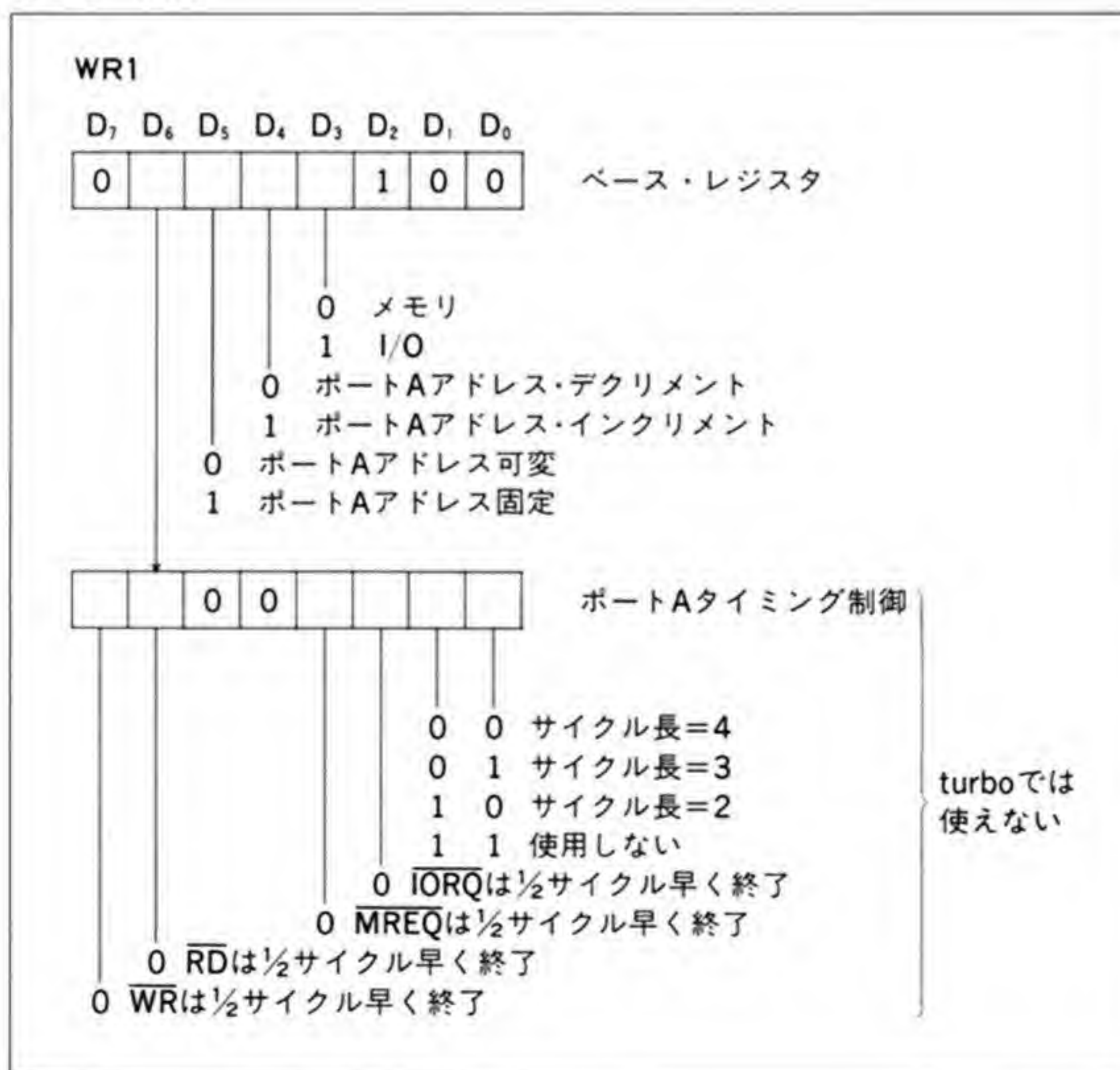
#### ⑥ $CD_H = \&B11001110$

WR4である。WR3はサーチ用なので、ここでは抜かしてある。 $D_3, D_2$ がポートB開始アドレスの指定。ここでは、 $00_H, 30_H$ がそうである。 $D_6, D_5$ が転送モードの指定である。それぞれの転送モードの詳細は次のとおり。

**バイトモード：**DMAは1バイト処理するたびにCPUにバスを返す。しかし、WR0で指定したブロックレングスだけのバイト数を処理し終わってなく、かつレディ信号（次にWR5で説明する）が有効であれば、その直後に再びバスを要求するので、CPUは1マシンサイクルしか命令を実行できない。早い話がDMAとCPUは交代でバスを使うことになる。turboではDMAを使ってフロッピーディスクにアクセスする場合、このモードを使っている（第9章を参照のこと）。

**バーストモード：**DMAはレディ信号が有効である間はCPUにバスを返さず、プログラム

図8-3 WR1





された処理が終わるまで動作を行ない続ける。レディ信号が無効になった場合はCPUにバスを返すが、再び有効になると、またバスを要求する。この章のサンプルはすべてこのモードを使っている。

**連続(コンティニユアス)モード**：これはDMAが我儘を通すモードで、DMAはプログラミングされたとおりの処理が終わるまで、レディ信号が無効であってもCPUにバスを返さない。つまりCPUはDMAが処理を終えるまで、何もできないことになる。特に素早い応答が要求されている場合に、このモードが使われる。

D<sub>4</sub>は割り込み制御バイトと呼ばれるサブレジスタを指すものである。サンプルでは使っ

図8-4 WR2

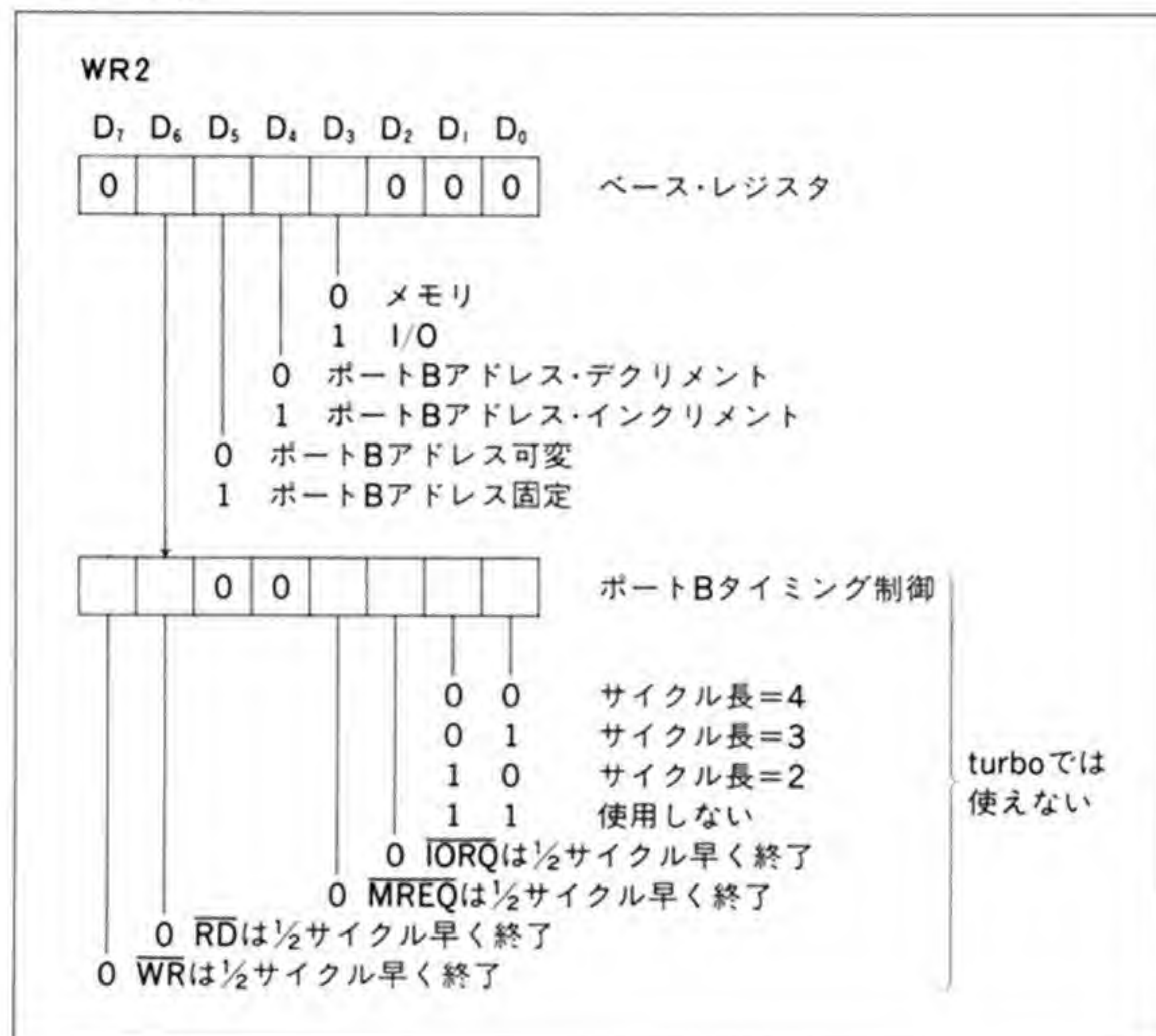
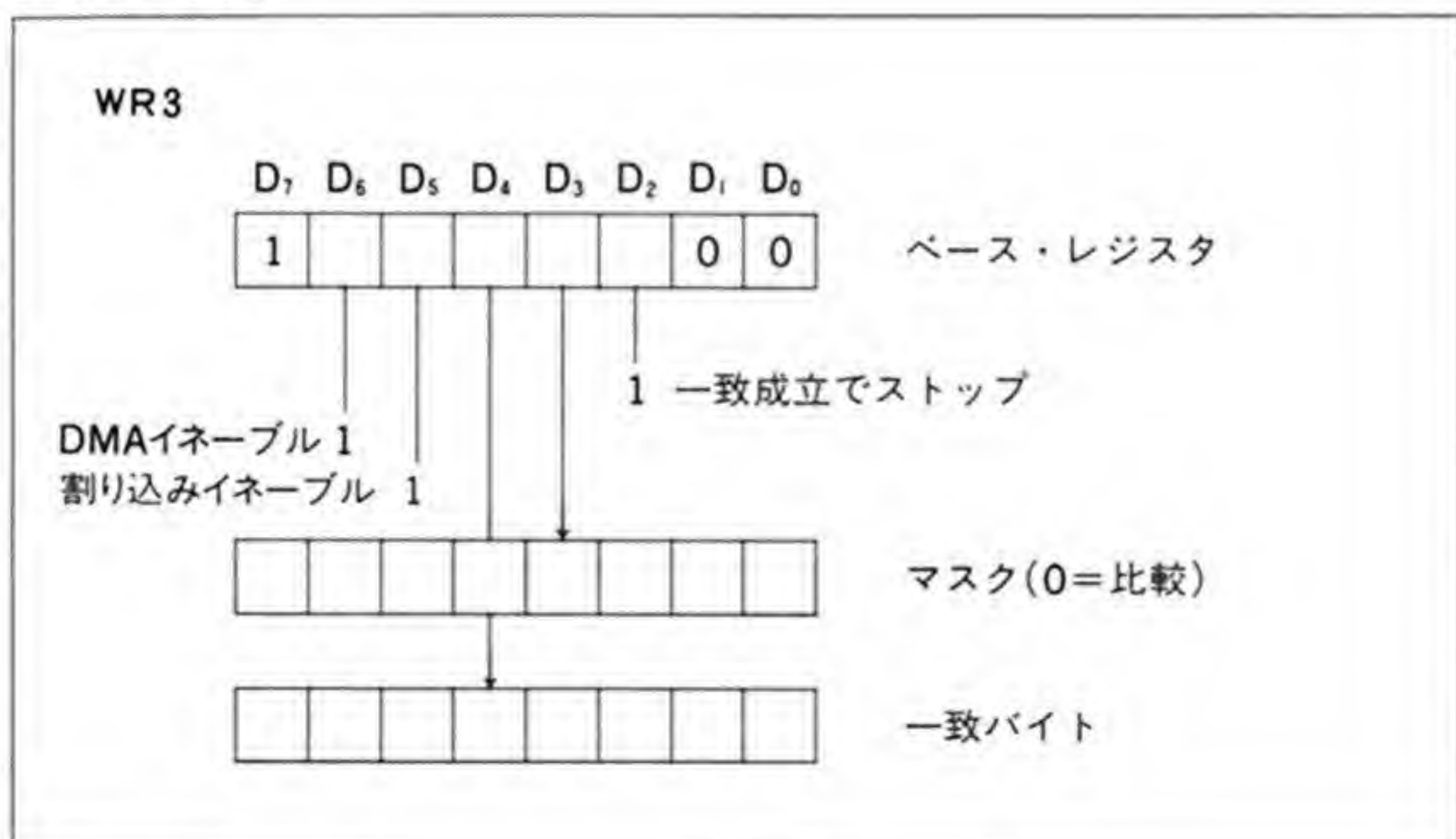


図8-5 WR3





ていないが一応説明しておく。まず、パルス発生(ビット2)は turbo では使えない。これは外部デバイスにパルスを送るものであるから、turbo にとっては必要なものではない。

次にインタラプト・オン (マッチ, エンド・オブ・ブロック, レディ) の三つであるが、これはそれぞれの条件のときに割り込みを起こせということ。ステータス・アフェクツ・ベクトルというのは割り込みベクトルを、割り込み要因ごとに变化させるかどうかのフラグである。パルス制御バイトは、パルス発生を使えないのであるから turbo では意味がない。割り込みベクトルは DMA から CPU に割り込みをかけるときのベクトルの下位アドレスを指定するもの。モード2の割り込みをきちんと理解してからいじるように (割

図 8-6 WR4

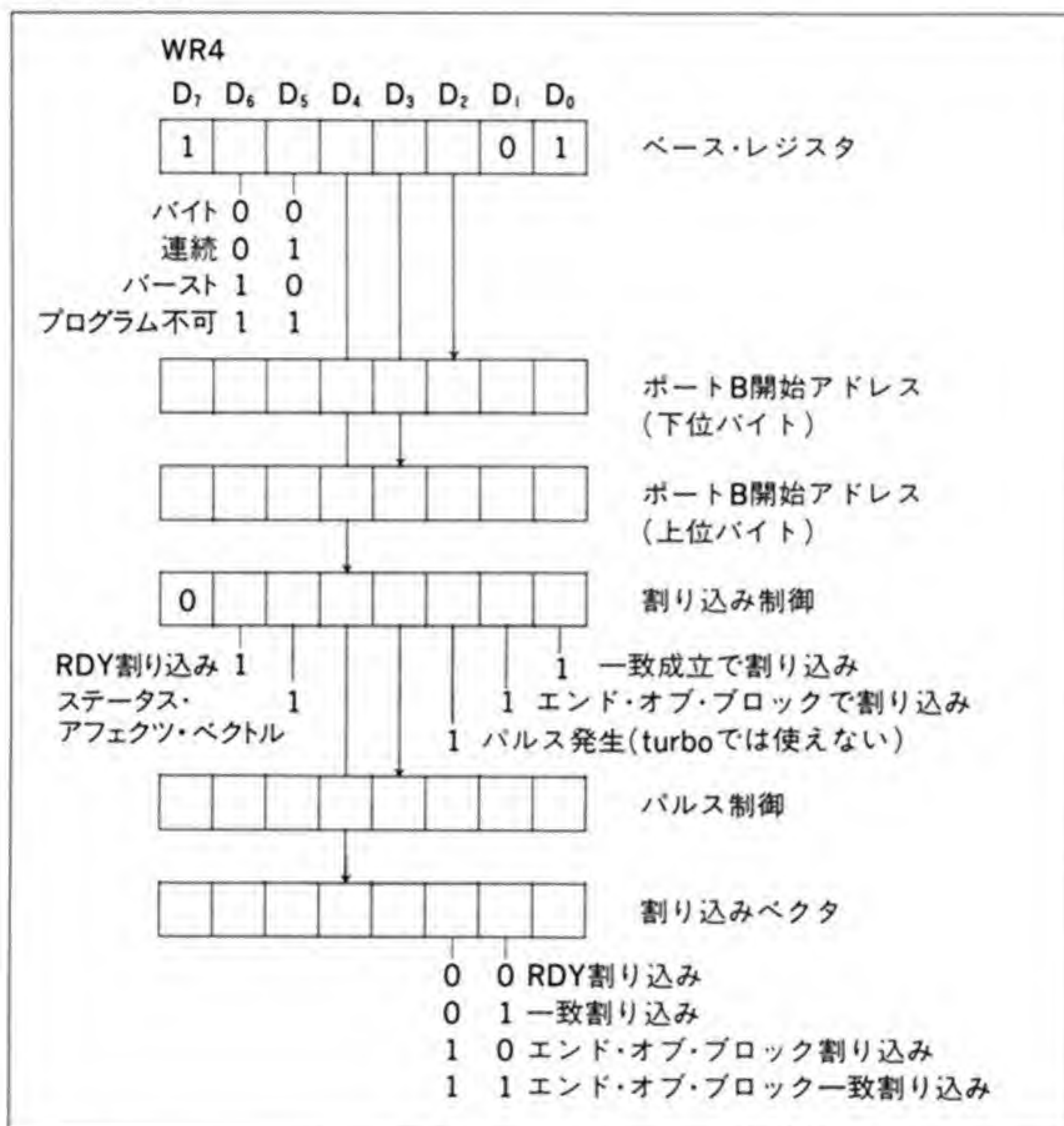
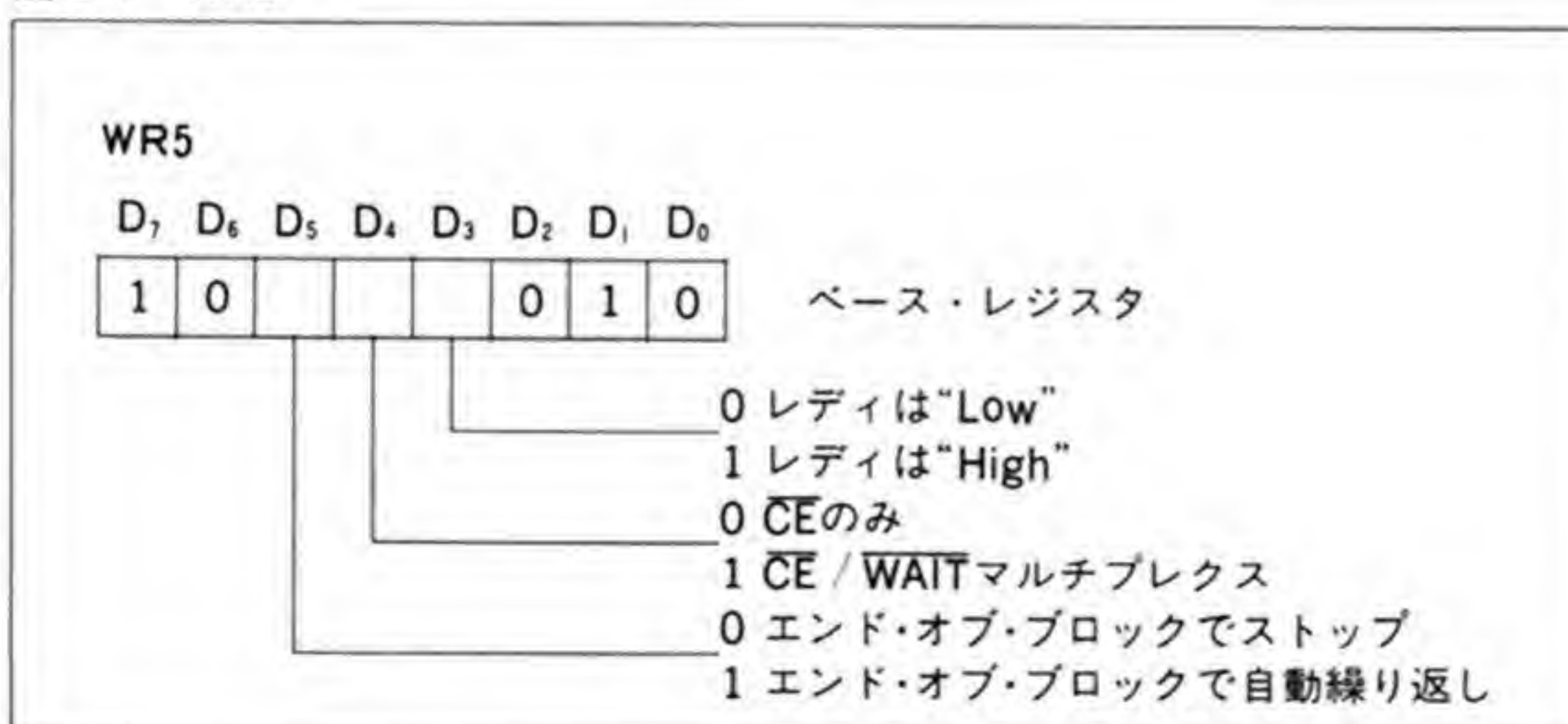


図 8-7 WR5





り込みに関しては第4章を参照のこと)。

# ⑦ 9A<sub>H</sub>=&B10011010

WR5である。まずD<sub>3</sub>であるが、DMAにはRDYという端子があって、そこへ入ってくる信号をレディ信号という。D<sub>3</sub>はレディ信号がHighのとき有効にするか、Lowのとき有効にするかの指定である。turboでは通常はこの信号はHighになっているので、このビットは1にしておく。逆に、FDCからデータ要求が来るとLowになるので、ディスクの読み書きをする場合は0にして、Lowを有効とする。D<sub>4</sub>は $\overline{\text{CE}}/\overline{\text{WAIT}}$ の使い方である。turboではこのビットを1にして $\overline{\text{WAIT}}$ を受け付けるようにすること。D<sub>5</sub>はオートリスタートの指定だが、これは0にして「エンド・オブ・ブロックで終決」にする。つまり、ブロックレングス分の処理を終えたらバスをCPUに返して止まるということである。もしオートリスタートにしたなら、レディ信号が有効である限り再スタートするので、止まってくれないことになる。

図8-8 リードレジスタの内容

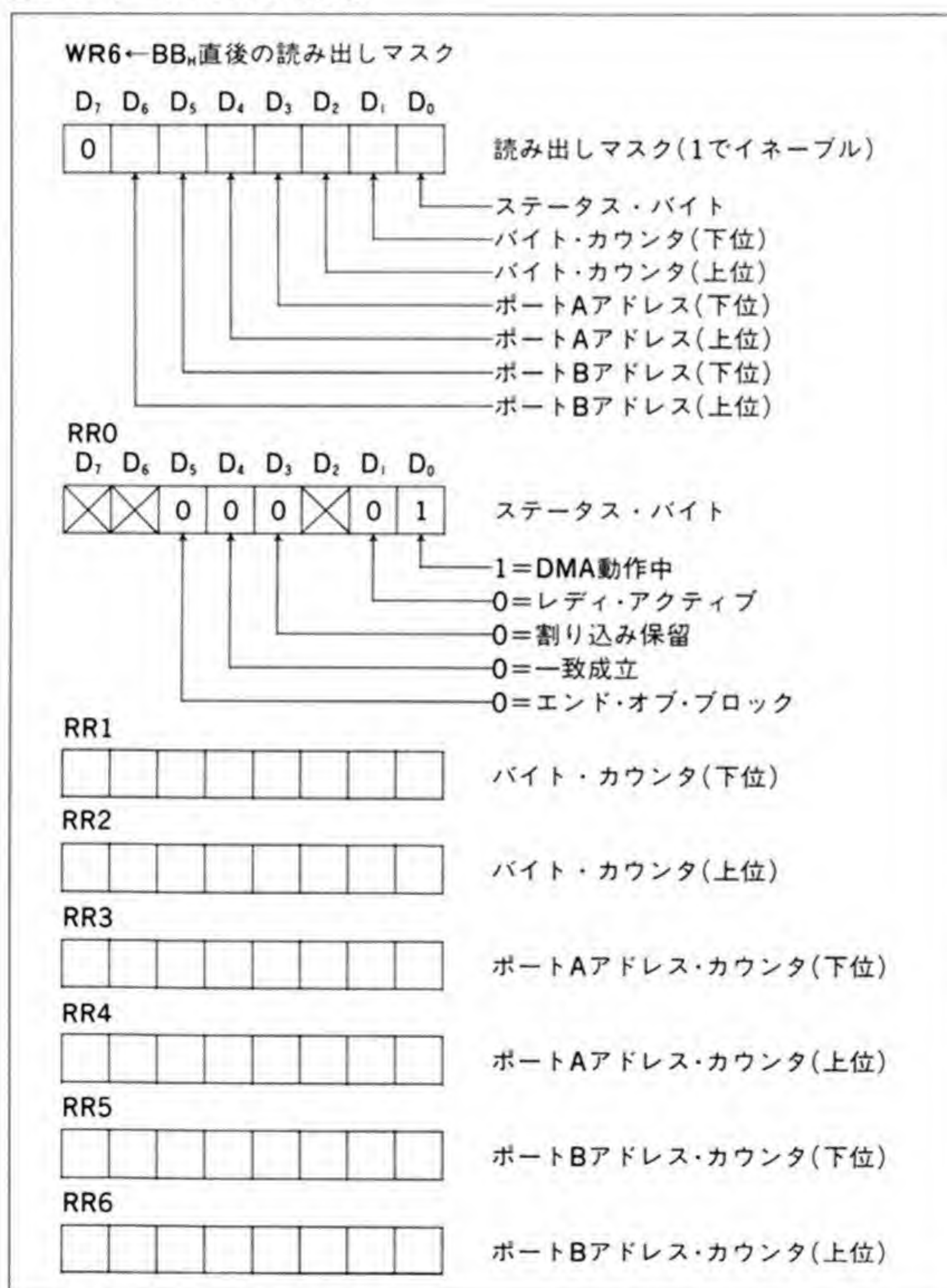




図 8-9 WR6

WR6							
D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>
1						1	1
83 <sub>H</sub>	ディセーブルコマンド。87 <sub>H</sub> (イネーブル)の逆。						
87 <sub>H</sub>	イネーブルコマンド。このコマンドを受け取ると、DMAは直ちにバス要求を出し動作を開始する。このコマンド以外のコマンドは、すべてDMAをディセーブルする。						
8B <sub>H</sub>	ステータス・バイト初期再設定コマンド。ステータス・バイトのビット 4(一致成立)および5(エンド・オブ・ブロック)を初期化する。						
A3 <sub>H</sub>	割り込みリセットおよびディセーブルコマンド。Z80を使っているturboには関係ない。8080などがRETIの代わりに使う。						
A7 <sub>H</sub>	読み出しシーケンス起動コマンド。このコマンドが送られると、マスク設定コマンド(BB <sub>H</sub> )で指定されたレジスタを順番に読み出せる。リードレジスタの図解を参照。						
AB <sub>H</sub>	割り込みイネーブルコマンド。電源投入時、DMAは割り込みディセーブル状態で起動するので、DMAの割り込み機能を使用するときはこのコマンドを送る。このコマンドをセットした後、割り込み条件が発生すると、DMAはCPUに対して割り込み要求を出し続けるので、割り込み処理ルーチンのはじめの方で割り込みディセーブルコマンド(AF <sub>H</sub> )を送ること。割り込み処理終了後、ふたたび割り込みイネーブルコマンドを送る前に、割り込み条件をリセットするためにステータスバイト初期再設定コマンド(8B <sub>H</sub> )を送る必要がある。						
AF <sub>H</sub>	割り込みディセーブルコマンド。DMAの割り込み処理ルーチンのはじめの方にこのコマンドが必要。そのルーチンの終わりには、ステータス・バイト初期再設定コマンド(8B <sub>H</sub> )と割り込みイネーブルコマンド(AB <sub>H</sub> )が必要。						
B3 <sub>H</sub>	強制RDYコマンド。このコマンドによって、RDY信号は強制的に常にアクティブであるとみなされる。このコマンドは、ロードコマンド(CF <sub>H</sub> )によってリセットされるので、ロードコマンドより後に送ってやる必要がある。また、バイトモードでは動作しない。						
B7 <sub>H</sub>	RETI後イネーブルコマンド。インタラプト・オン・レディ・モードのときのみ使用される(WR4、割り込み制御バイトのビット6)。このモードでRDY信号がアクティブになるとDMAは割り込み要求を出す。割り込み処理後、DMAにバス要求を出させたい場合、RETI後イネーブルコマンド(B7 <sub>H</sub> )、DMAイネーブルコマンド(87 <sub>H</sub> )を送った後、RETI命令を実行すればよい。						
BB <sub>H</sub>	読み出しマスク設定コマンド。このコマンドに続く1バイトデータは、読み出しマスクレジスタへセットされる。マスクでセットされた読み出しレジスタの読み出しシーケンスは、コマンドA7 <sub>H</sub> によって起動する。						
BF <sub>H</sub>	ステータス・バイト読み出しコマンド。ステータス・バイト(1バイト)だけを読む場合のコマンド。						
C3 <sub>H</sub>	リセット・コマンド。電源投入時やDMAのプログラム打ち切り時に使用される。						
C7 <sub>H</sub>	WR1のポートA可変タイミグを標準Z80タイミグにリセットする(コマンドC3 <sub>H</sub> にもこの機能がある。ただし、turboでは可変タイミグは使えない)。						
CB <sub>H</sub>	WR2のポートB可変タイミグのリセット(同上)。						
CF <sub>H</sub>	ロードコマンド。ソース、ディスティネーションのアドレスレジスタの内容をアドレスカウンタにロードする。同時に、バイトカウンタのクリアも行なう。このコマンドによって、ソース側のアドレスは直ちにカウンタにロードされるが、ディスティネーション側のアドレスは、最初のインクリメントorデクリメントの際にロードされるのでディスティネーション・アドレスが固定の場合は、特別な操作を必要とする。また、このコマンドは、強制RDYコマンド(B3 <sub>H</sub> )およびステータス・バイトのビット0(DMA動作中)をリセットしてしまう。						
D3 <sub>H</sub>	コンティニューコマンド。バイトカウンタをクリアするが、両ポートのアドレスカウンタはロードせずに現状の値を保持する。						



⑧ CF<sub>H</sub>=&B11001111

WR6である。WR6はビットごとに意味を持つわけではない。CF<sub>H</sub>はロードコマンドであるが、これは設定された値をワーク用のカウンタにロードするものである。早い話が、「よーい、どん！」の「よーい」である。

⑨ B3<sub>H</sub>=&B10110011

強制レディコマンド。つまり、RDY信号を常に有効と見なせ、というコマンド。WR5の9A<sub>H</sub>でHighを有効にしているから、本当は必要ない。しかし、RDY信号が関係なくて、とにかくブロックレンゲス分の処理をさせたい場合には入れておく方がよい。「よーい、どん！」の「よーい」と「どん！」の間に、「途中で止まるな」と耳元にささやくよーいなるものである。

⑩ 87<sub>H</sub>=&B10000111

WR6で、イネーブルコマンド。要するに「どん！」で、DMAはこのコマンドを受け取ると、それつとばかりに動き出す。DMAが動き出すと、バスの管理はCPUを離れてDMAに移る。こうなるとCPUは命令を読み出すこと（つまりフェッチ）もできない。すなわち金縛りにあうわけである。だからZ80のプログラムとしては、続けて何事もなかったかのように「転送直後にすること」が書かれていけばよいことになる。

以上がこのサンプルの解説である。はっきり言って異常に面倒臭いが、それは最初のうちだけで、一度コマンド列を作ってしまうと、やることを変えるのに、それほど手間はかからない。

そこで、基本的なバリエーションを示してしまえばよい、というわけで、突如としてサンプルプログラムが山津波を起こすのであった。

A) リスト 8-5 (正しい手抜き法)

DMAでは、設定し直さなければ、以前に設定したライトレジスタとサブレジスタを使うことになる。つまり、似たような処理を続けてDMAに行なわせる場合、当然レジスタの設定も似てくるから、いちいち再設定しなくてもよいレジスタが出てくる。リスト 8-5は、そこ極端な例で、WR6以外は同じままでよいので、150～190行の間のループではソースへASCIIコードをPOKEした後、CF<sub>H</sub>=ロード、B3<sub>H</sub>=強制レディ、87<sub>H</sub>=イネーブルだけをDMAに送っている。変化しているのはソースアドレスではなく、ソースアドレスの内容であるからWR0のポートA開始アドレスは再設定する必要がない。やっていることは、リスト 8-3と同じく、0～255のASCIIコードの文字を画面中に書いているだけである。ループの中ではDMAに送るのは3バイト（本当はB3<sub>H</sub>も抜かしてよい）だけになったので、結構速くなっている。

リスト 8-5 正しい手抜き法

```
100 CLEAR&HFFFF
110 RESET$=HEXCHR$("C3 C3 C3 C3 C3 C3")
120 DMA$=RESET$+HEXCHR$("7D 00 E0 D0 07 24 18 CD 00 30 9A")
130 GOSUB"SETDMA"
```



```

140 '
150 FORK=0TO255
160 POKE&HE000,K:'アスキーコードをソース・アドレスへPOKE。
170 DMA$=HEXCHR$("CF B3 87"):'←これだけです、これだけ。
180 GOSUB"SETDMA"
190 NEXT
200 END
210 '
220 LABEL"SETDMA"
230 FORI=1TOLEN(DMA$):OUT&H1F80,ASC(MID$(DMA$,I,1)):NEXT
240 RETURN

```

## B) リスト 8-6 (メモリ→メモリ)

単純なブロック転送である。たかだか 64 バイトの転送なら、MEM\$(~)=MEM\$(~)で充分なのであるが、結局サンプルとはそういうものなのである。40 行と 90 行でモニタに飛んでメモリをダンプしている。それ以外に特筆することはない。

リスト 8-6 メモリ→メモリのブロック転送

```

10 CLEAR &HFFFF:CLS4
20 MEM$(&HE000,&H90)=STRING$(&H90,0) : 'メモリをクリア
30 MEM$(&HE000,&H40)=STRING$(&H40,"!"): 'ソースをセット
40 KEY0,"DE000 E08F"+CHR$(13)+"R"+CHR$(13):MON
50 A$=INKEY$(1)
60 RESET$=HEXCHR$("C3 C3 C3 C3 C3 C3"): 'リセット データ
70 DMA$=RESET$+HEXCHR$("7D 00 E0 3F 00 14 10 CD 40 E0 9A CF B3 87")
80 GOSUB"SETDMA"
90 KEY0,"DE000 E08F"+CHR$(13)+"R"+CHR$(13):MON
100 END
110 '
120 LABEL"SETDMA"
130 FORI=1TOLEN(DMA$):OUT&H1F80,ASC(MID$(DMA$,I,1)):NEXT
140 RETURN

```

## C) リスト 8-7 (メモリ→I/O)

「CLS 4」と大体同じことをするプログラムである。具体的には、テキスト VRAM へ 32(スペースの ASCII コード), アトリビュートへ 7, 漢字 VRAM とグラフィック VRAM へ 0 を書き込む。ただしグラフィックの方はバンク 0 か 1, どちらか一方, アクセス可能な状態になっているバンクだけをクリアする。気になったので 100 回実行して「CLS 4」と速度を比べてみた。約 2 倍の速度であった。140 行の HEXCHR\$の中には機械語ルーチンである。ソースリストはリスト 8-8 である。OUTI を使ったテクニックで少しでも速くしようということをもくろんでいる (本当は JR を JP にするともっと速くなる)。USR0 (DMA\$)を実行すると, DE レジスタに DMA\$の内容へのポインタ, B レジスタに文字列の長さが入って E000<sub>H</sub>にサブルーチンコールするのである。また, 230 行の 0, 32, 7, は転送するデータなので, 0→2 などのように変えると 5 分間ぐらひは楽しめるであろう (100 回ループは消すんだよ)。

リスト 8-7 「CLS 4」モドキ

```

100 DEFINT A-Z
110 INIT:KLIST0:CONSOLE0,25

```



```

120 CLEAR&HDFFF
130 DEFUSR0=&HE000
140 MEM$(&HE000,12)=HEXCHR$("78 EB 01 80 1F 04 ED A3 3D 20 FA C9")
150 '機械語ルーチンを作って速くしたのだ。
160 RESET$=HEXCHR$("C3 C3 C3 C3 C3 C3"):'リセット データ
170 DMA$=RESET$+HEXCHR$("7D 20 E0 CF C7 24 18 CD 00 38 9A CF 87")
180 '漢字 V R A M とグラフィック V R A M をクリア。
190 DMA$=DMA$+HEXCHR$("4D 21 07 C9 30 CF 87")
200 'テキスト V R A M を 3 2 (スペース) に。
210 DMA$=DMA$+HEXCHR$("0D 22 C9 28 CF 87")
220 'アトリビュートを 7 (白) に。
230 POKE&HE020,0,32,7:'それぞれのデータです。
240 TIME=0:FORK=1TO100:D$=USR0(DMA$):NEXT:T1=TIME
250 TIME=0:FORK=1TO100:CLS4:NEXT:T2=TIME
260 PRINTT1,T2
270 END

```

リスト 8-8 高速版“SETDMA”

```

                                .Z80
                                .PHASE 0E000H ;OR ANY PLACE
                                ;
                                ;DE=ADDRESS TO DATA,B=COUNTER
E000 78 LD A,B
E001 EB EX DE,HL ;HL POINTS DATA
E002 01 1F80 LD BC,1F80H ;DMA ADDRESS
E005 04 LOOP: INC B
E006 ED A3 OUTI
E008 3D DEC A
E009 20 FA JR NZ,LOOP
                                ;
E00B C9 RET
                                ;
                                END

```

#### D) リスト 8-9 (I/O → I/O)

DMA のありがたみは、やはり LDIR が使えず、なおかつポインタとして BC レジスタしか使えない I/O へのアクセス時に七色するのである。最初はテキスト VRAM → テキスト VRAM の転送である。110 行にあるどうしようもない文字列を 0 行目にプリントした後、同順、逆順に転送する。アトリビュートや漢字 VRAM には触っていないので、何かに使おうとする場合は注意が必要である。3 とおりのコマンド列を作っておいたから、「」(REM) を消して使っていただきたい。特に 180 行の「同じ行内へ逆順に転送する」というやつは、ソースとディスティネーションが重なるとどうなるかという意味も含んでいる(結果は別にどうということもないが)。

リスト 8-9 I/O → I/O(文字列)転送

```

100 CLS4
110 PRINT"0123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmn"
120 PRINTSTRING$(160," ")
130 RESET$=HEXCHR$("C3 C3 C3 C3 C3 C3"):'リセット データ
140 '
150 '行を替えて同順
160 DMA$=RESET$+HEXCHR$("7D 00 30 4F 00 1C 18 CD 50 30 9A CF B3 87")
170 '同じ行で逆順
180 'DMA$=RESET$+HEXCHR$("7D 00 30 4F 00 1C 08 CD 4F 30 9A CF B3 87")
190 '行を替えて逆順
200 'DMA$=RESET$+HEXCHR$("7D 00 30 4F 00 1C 08 CD 9F 30 9A CF B3 87")
210 '
220 GOSUB"SETDMA"

```



```

230 LOCATE 0,10
240 END
250 '
260 LABEL"SETDMA"
270   FOR I=1 TO LEN(DMA$):OUT&H1F80,ASC(MID$(DMA$,I,1)):NEXT
280 RETURN

```

#### E) リスト 8-10 (I/O → I/O)

人呼んで「回転星」という。このプログラムは EMM（大容量 RAM ボード）が必要だから、持っていない人は指をくわえて見ていていただきたい。

このプログラムが何をしているかというと、EMM ↔ G-RAM の間でデータ転送をしているのである。グラフィックデータはモノクロで 1 ページ当たり  $640 \times 200 = 16K$  バイトである、EMM の容量は 320K バイトだから、計算すれば  $320 \div 16 = 20$  画面分が 1 枚の EMM に入ってしまう。8 色なら  $320/48 = 6.66\cdots$  であるから 6 画面分。640×400 ならばその半分の 3 画面分である。

#### リスト 8-10 回転星

```

100 WIDTH80,25:INIT:KLIST0:CLS4
110 OUT&HD00,0:OUT&HD01,0:OUT&HD02,0:' E M M の内部アドレスを 0 にする。
120 'データを E M M に送りこむ。( 20 面分 )
130 S=360/20/5:FORJ=1TO20
140 POLY(320,100),90,4,144,90+J*S,720+90+J*S:' 星形を書く
150 PAINT(0,0),2,4:PAINT(320,100),1,2
160 GOSUB"EMMW":CLS4
170 NEXT:BEEP
180 '
190 INIT:CLS4:F=0:Q=0:B=0
200 OUT&HD00,0:OUT&HD01,0:OUT&HD02,0:' E M M の内部アドレスを 0 にする。
210 Q=((Q+1)AND7):B=((Q+1)AND7)
220 FORJ=1TO20
230 IF 0 GOSUB"EMMR" ELSE GOSUB"EMMR-2":' 次々と G R A M へ転送する。
240 NEXT:GOTO 200
250 END
260 '
270 LABEL"EMMW":' G R A M から E M M へ転送する。
280 DMA$=HEXCHR$("C3C3C3 C3 C3 C3 7D 03 0D FF 3F 2C 18 CD 00 40 9A CF 01 CF 87")
290 GOSUB"SETDMA"
300 RETURN
310 '
320 LABEL"EMMR":' E M M から G R A M へ転送する。
330 DMA$=HEXCHR$("C3 C3 C3 C3 C3 C3 7D 03 0D FF 3F 2C 18 CD 00 40 9A CF B3 87")
340 GOSUB"SETDMA"
350 RETURN
360 '
370 LABEL"EMMR-2":' 画面スワッピングも使いながら E M M から G R A M へ転送する。
380 IF F THEN 420
390 DMA$=HEXCHR$("C3 C3 C3 C3 C3 C3 7D 03 0D FF 3F 2C 18 CD 00 40 9A CF B3 87")
400 GOSUB"SETDMA":PALET@ B,Q,B,Q,B,B,B,B
410 GOTO 440
420 DMA$=HEXCHR$("C3 C3 C3 C3 C3 C3 7D 03 0D FF 3F 2C 18 CD 00 80 9A CF B3 87")
430 GOSUB"SETDMA":PALET@ B,B,Q,Q,B,B,B,B
440 F=(F=0):RETURN
450 '
460 LABEL"SETDMA"
470   FOR I=1 TO LEN(DMA$):OUT &H1F80,ASC(MID$(DMA$,I,1)):NEXT
480 RETURN

```

このプログラムは DMA を使い、EMM から G-RAM へデータを転送することにより、一気にアニメーションしてしまおうというものである。ここでは星を回しているだけだが、どんな図形だろうと同じ手間だから、結構面白いことができるはずである。

というところで、リスト 8-10 の解説を行なう。130～170 行が星型のグラフィックを描き、EMM へ送り込むルーチンである。EMM の内部アドレスは自動的にインクリメントするから、110 行と 200 行で EMM の内部アドレスを設定するだけで済む。



さて、実はこのサンプルは、

**とっても大事なのである。**

というのは、DMA ではディスティネーションが**固定アドレスの場合は特別な操作が必要**なのである。EMM へのデータ書き込みは、固定アドレスへの転送だから、これにひっかるのである。具体的には、ディスティネーションに使用されるポート（この場合はポート A）に実際にアドレスがロードされるのは、最初の増減（インクリメント/デクリメント）が行なわれたときなのだ。だからアドレス固定の場合は死ぬまでロードされない。そのために、少々手間をかけてやる必要がある。早い話が、とりあえず一度ポート A を**ソースに指定してから** CF<sub>H</sub>（ロードコマンド）を送ってやるのである。ソースの方は、ロードコマンドが送られると、その時点でちゃんとアドレスがロードされるから OK。その直後に「やっぱりポート A はディスティネーションにするね」と、ひっくり返すのである。280 行にある HEXCHR\$ の中の後ろの方の「CF 01」がその部分である。01<sub>H</sub>（WR0：B → A 指定）が鍵なのである。その後は「CF 87」で再ロード、DMA イネーブルで走ってくれる。

次に 230 行の「IF 0～」というやつであるが、これは 2 とおり用意した EMM → G-RAM 転送のうち、どちらを選ぶかの分岐点になっている。GOSUB “EMMR” の方がシンプルだが、転送している様子が見えてしまうのであまり美しくない。そこで “EMMR-2” では青画面（もちろん色はパレットで変えてある）を表示しつつ、赤画面へ転送し、転送後青画面を引っ込めて、赤画面を表示、次は赤画面と青画面の役割を入れ替えて同じことを繰り返す、ということをやっている。どちらもそれほど速くないが、これは DMA のせいではなく、460 行からのサブルーチン “SETDMA” がトロいのである。リスト 8-8 の機械語サブルーチンを使えばほぼ 2 倍の速度になる。リスト 8-11、8-12 に DMA を使わずに CPU がシコシコと転送するプログラムも載せておくので、turbo ユーザーでない人も試して欲しい。ただし NEW BASIC (CZ-8F01 V 2.0) でないとペイントにいらだつと思われるので覚悟するよーに。

なお、リスト 8-11 で USR0, USR1 に文字列を渡しているのは、&H8000 が実数型になってしまうためである。さらに補足すると、リスト 8-12 のそれぞれの先頭にある「CP 3」は A レジスタが渡された変数のタイプを持っているので、それをチェックしているのである。文字列型以外であれば、タイプミスマッチのエラーが発生するようにしてある。肝心の速度の比較であるが、リスト 8-10 とたいして変わらないようである。

#### リスト 8-11 DMA を使わない回転星

```
100 CLEAR&HFFFF
110 DEFUSR0=&HE000:DEFUSR1=&HE030
120 MEM$(&HE000,16)=HEXCHR$("FE 03 28 04 3E 0D DD E9 EB 46 23 4E 21 00 40 D9")
130 MEM$(&HE010,16)=HEXCHR$("01 03 0D D9 ED 78 03 D9 ED 79 D9 2B 7C B5 C2 14")
140 MEM$(&HE020,2)=HEXCHR$("E0 C9")
150 MEM$(&HE030,16)=HEXCHR$("FE 03 28 04 3E 0D DD E9 EB 46 23 4E D9 21 00 40")
160 MEM$(&HE040,16)=HEXCHR$("01 03 0D ED 78 D9 ED 79 03 D9 2B 7C B5 C2 43 E0")
170 MEM$(&HE050,1)=HEXCHR$("C9")
180 WIDTH80:INIT:CLS4
190 OUT&HD00,0:OUT&HD01,0:OUT&HD02,0:' E M M の内部アドレスを 0 にする。
200 'データを E M M に送りこむ。(20面分)
210 S=360/20/5:FORJ=1TO20
220 POLY(320,100),90,4,144,90+J*S,720+90+J*S:'星形を書く
230 PAINT(0,0),2,4:PAINT(320,100),1,2
240 GOSUB"EMMW-NM":CLS4
```



```

250 NEXT:BEEP
260 '
270 INIT:CLS4:F=0:Q=0:B=0
280 OUT&HD00,0:OUT&HD01,0:OUT&HD02,0:' E M M の内部アドレスを0にする。
290 Q=((Q+1)AND7):B=((Q+1)AND7)
300 FORJ=1TO20
310 IF 0 GOSUB"EMMR-NM" ELSE GOSUB"EMMR-NM-2":' 次々と G R A M へ転送する。
320 NEXT:GOTO 280
330 END
340 '
350 LABEL"EMMW-NM":' G R A M から E M M へ転送する。
360 D$=USR0(HEXCHR$( "4000" ))
370 RETURN
380 '
390 LABEL"EMMR-NM":' E M M から G R A M へ転送する。
400 D$=USR1(HEXCHR$( "4000" ))
410 RETURN
420 '
430 LABEL"EMMR-NM-2":' 画面スワッピングも使いながら E M M から G R A M へ転送する。
440 IF F THEN 480
450 D$=USR1(HEXCHR$( "4000" ))
460 PALET0,B:PALET1,Q:PALET2,B:PALET3,Q
470 GOTO 500
480 D$=USR1(HEXCHR$( "8000" ))
490 PALET0,B:PALET1,B:PALET2,Q:PALET3,Q
500 F=(F=0):RETURN

```

リスト 8-12 EMM↔G-RAM サブルーチン

			.PHASE	0E000H
			.Z80	
		;STAR		
E000	FE 03		CP	3 ;CHECK VAR.TYPE
E002	28 04		JR	Z,OKGO1 ;HL POINTS.INT.?
E004	3E 0D		LD	A,13 ;TYPE MISMATCH
E006	DD E9		JP	(IX) ;ERROR ROUTINE
		;OKGO1:		
E008	EB		EX	DE,HL
E009	46		LD	B,(HL) ;HIGH
E00A	23		INC	HL
E00B	4E		LD	C,(HL) ;LOW
E00C	21 4000		LD	HL,4000H ;COUNTER
E00F	D9		EXX	
E010	01 0D03		LD	BC,0D03H ;EMM DATA
		;LOOP1:		
E013	D9		EXX	
E014	ED 78		IN	A,(C) ;FM GRAM
E016	03		INC	BC
E017	D9		EXX	
E018	ED 79		OUT	(C),A
E01A	D9		EXX	
E01B	2B		DEC	HL
E01C	7C		LD	A,H
E01D	B5		OR	L
E01E	C2 E014		JP	NZ,LOOP1
		;RET		
E021	C9		RET	
		;DASAI		
E022			DS	14 ;DASAI
		;OKGO2:		
E030	FE 03		CP	3 ;CHECK VAR.TYPE
E032	28 04		JR	Z,OKGO2 ;HL POINTS.INT.?
E034	3E 0D		LD	A,13 ;TYPE MISMATCH
E036	DD E9		JP	(IX) ;ERROR ROUTINE
		;OKGO2:		
E038	EB		EX	DE,HL
E039	46		LD	B,(HL) ;HIGH
E03A	23		INC	HL
E03B	4E		LD	C,(HL) ;LOW



E03C	D9	EXX		
E03D	21 4000	LD	HL,4000H	;COUNTER
E040	01 0D03	LD	BC,0D03H	;EMM DATA
E043	ED 78	LOOP2: IN	A,(C)	;FM EMM
E045	D9	EXX		
E046	ED 79	OUT	(C),A	
E048	03	INC	BC	
E049	D9	EXX		
E04A	2B	DEC	HL	
E04B	7C	LD	A,H	
E04C	B5	OR	L	
E04D	C2 E043	JP	NZ,LOOP2	
E050	C9	RET		
		END		

#### F) リスト 8-13, 8-14 (テキストスクロール)

別に言うことはないが、このプログラムを走らせると、画面に表示されている漢字が一瞬化ける。これは三つの転送(テキスト VRAM, アトリビュート, 漢字 VRAM)の間に時間差があるため。つまり“SETDMA”が遅いからである。

##### リスト 8-13 テキストスクロールアップ

```

10 RESET$=HEXCHR$("C3 C3 C3 C3 C3 C3"): 'リセット データ
20 'テキスト V R A M
30 DMA$=RESET$+HEXCHR$("7D 50 30 7F 07 1C 18 CD 00 30 9A CF B3 87")
40 'アトリビュート
50 DMA$=DMA$ +HEXCHR$("7D 50 28 7F 07 1C 18 CD 00 28 9A CF B3 87")
60 '漢字 V R A M
70 DMA$=DMA$ +HEXCHR$("7D 50 38 7F 07 1C 18 CD 00 38 9A CF B3 87")
80 GOSUB"SETDMA"
90 END
100 '
110 LABEL"SETDMA"
120 FOR I=1 TO LEN(DMA$): OUT&H1F80,ASC(MID$(DMA$,I,1)):NEXT
130 RETURN

```

##### リスト 8-14 テキストスクロールダウン

```

10 RESET$=HEXCHR$("C3 C3 C3 C3 C3 C3"): 'リセット データ
20 'テキスト V R A M
30 DMA$=RESET$+HEXCHR$("7D 7F 37 7F 07 0C 08 CD CF 37 9A CF B3 87")
40 'アトリビュート
50 DMA$=DMA$ +HEXCHR$("7D 7F 2F 7F 07 0C 08 CD CF 2F 9A CF B3 87")
60 '漢字 V R A M
70 DMA$=DMA$ +HEXCHR$("7D 7F 3F 7F 07 0C 08 CD CF 3F 9A CF B3 87")
80 GOSUB"SETDMA"
90 END
100 '
110 LABEL"SETDMA"
120 FOR I=1 TO LEN(DMA$): OUT&H1F80,ASC(MID$(DMA$,I,1)):NEXT
130 RETURN

```

#### G) リスト 8-15 (上8ドットグラフィックススクロール)

これは8ドット分グラフィックスを上へスクロールさせるプログラムである。

4050<sub>H</sub>~477F<sub>H</sub> → 4000<sub>H</sub>~472F<sub>H</sub>



4850<sub>H</sub>~4F7F<sub>H</sub>→4800<sub>H</sub>~4F2F<sub>H</sub>

⋮  
⋮  
⋮

7850<sub>H</sub>~7F7F<sub>H</sub>→7800<sub>H</sub>~7F2F<sub>H</sub>

という具合に8回ブロック転送を実行している。これはX1のグラフィックの座標とアドレスの関係からこのようになっているのである。このプログラムではY座標が192~199に当たる下の方の8ライン分はそのまま残るので、140行のサークル命令の半径を95ぐらいにするとゴミが出る。

リスト 8-15 上8ドットグラフィックスクロール

```
100 DEFINT A-Z
110 CLEAR&HFFFF
120 DEFUSR0=&HE000
130 MEM$(&HE000,12)=HEXCHR$("78 EB 01 80 1F 04 ED A3 3D 20 FA C9")
140 INIT:CLS4:CIRCLE(320,100),60,6
150 PAINT(320,100),&H26,6
160 SYMBOL(282,100),"朝です",2,2,7,0,PRESET
170 '
180 RESET$=HEXCHR$("C3 C3 C3 C3 C3 C3"): 'リセット データ
190 '青画面
200 DMA$=RESET$+HEXCHR$("7D 50 40 7F 07 1C 18 CD 00 40 9A CF 87")
210 DMA$=DMA$+HEXCHR$("15 48 C9 48 9A CF 87")
220 DMA$=DMA$+HEXCHR$("15 50 C9 50 9A CF 87")
230 DMA$=DMA$+HEXCHR$("15 58 C9 58 9A CF 87")
240 DMA$=DMA$+HEXCHR$("15 60 C9 60 9A CF 87")
250 DMA$=DMA$+HEXCHR$("15 68 C9 68 9A CF 87")
260 DMA$=DMA$+HEXCHR$("15 70 C9 70 9A CF 87")
270 DMA$=DMA$+HEXCHR$("15 78 C9 78 9A CF 87")
280 '赤画面
290 DMA$=DMA$+HEXCHR$("15 80 C9 80 9A CF 87")
300 DMA$=DMA$+HEXCHR$("15 88 C9 88 9A CF 87")
310 DMA$=DMA$+HEXCHR$("15 90 C9 90 9A CF 87")
320 DMA$=DMA$+HEXCHR$("15 98 C9 98 9A CF 87")
330 DMA$=DMA$+HEXCHR$("15 A0 C9 A0 9A CF 87")
340 DMA$=DMA$+HEXCHR$("15 A8 C9 A8 9A CF 87")
350 DMA$=DMA$+HEXCHR$("15 B0 C9 B0 9A CF 87")
360 DMA$=DMA$+HEXCHR$("15 B8 C9 B8 9A CF 87")
370 '緑画面
380 DMA$=DMA$+HEXCHR$("15 C0 C9 C0 9A CF 87")
390 DMA$=DMA$+HEXCHR$("15 C8 C9 C8 9A CF 87")
400 DMA$=DMA$+HEXCHR$("15 D0 C9 D0 9A CF 87")
410 DMA$=DMA$+HEXCHR$("15 D8 C9 D8 9A CF 87")
420 DMA$=DMA$+HEXCHR$("15 E0 C9 E0 9A CF 87")
430 DMA$=DMA$+HEXCHR$("15 E8 C9 E8 9A CF 87")
440 DMA$=DMA$+HEXCHR$("15 F0 C9 F0 9A CF 87")
450 DMA$=DMA$+HEXCHR$("15 F8 C9 F8 9A CF 87")
460 FORK=0 TO 20: D$=USR0(DMA$): NEXT
470 END
```

#### H) リスト 8-16 (左8ドットグラフィックスクロール)

4001<sub>H</sub>~7FFF<sub>H</sub>→4000<sub>H</sub>~7FFE<sub>H</sub>

8001<sub>H</sub>~BFFF<sub>H</sub>→8000<sub>H</sub>~BFFE<sub>H</sub>

C001<sub>H</sub>~FFFF<sub>H</sub>→C000<sub>H</sub>~FFFE<sub>H</sub>

の3回ブロック転送をしている。考えられる限りセコいので、80回(640ドット)左ヘスクロールすると8ドット分上へ上がってしまう。これについては、普通の機械語でも試し



て速度を比べてみた。270 行からがそうである。USR1 に「8 ドット左ヘスクロールする回数」を渡せばよい。サンプル中では 80 を渡してある。機械語部分のソースはリスト 8-17 である。本当はありとあらゆる汚い手を使って高速化してみたかったのだが、読みづらくなるので避けたのである。もっとも、このサンプルのように、BC と B'C' をソースとディスティネーションに割り当てるのが一番速いと思う。

リスト 8-16 左 8 ドットグラフィックススクロール

```

100 DEFINT A-Z
110 CLEAR&HDEFF
120 DEFUSR0=&HE000:DEFUSR1=&HE100
130 MEM$(&HE000,12)=HEXCHR$("78 EB 01 80 1F 04 ED A3 3D 20 FA C9")
140 INIT:CLS4:CIRCLE(320,100),60,6
150 PAINT(320,100),&H26,6
160 SYMBOL(266,100),"異常です",2,2,7,0,PRESET
170 '
180 RESET$=HEXCHR$("C3 C3 C3 C3 C3 C3"): 'リセット データ
190 ' 青画面
200 DMA$=RESET$+HEXCHR$("7D 01 40 FE 3F 1C 18 CD 00 40 9A CF 87")
210 ' 赤画面
220 DMA$=DMA$ +HEXCHR$("15      80              C9      80 9A CF 87")
230 ' 緑画面
240 DMA$=DMA$ +HEXCHR$("15      C0              C9      C0 9A CF 87")
250 TIME=0:FORK=0 TO 80:D$=USR0(DMA$):NEXT:PRINT"BY DMA   =":TIME
260 '
270 MEM$(&HE100,16)=HEXCHR$("FE 02 28 04 3E 0D DD E9 7E B7 C8 57 01 00 40 CD")
280 MEM$(&HE110,16)=HEXCHR$("23 E1 01 00 80 CD 23 E1 01 00 C0 CD 23 E1 15 C2")
290 MEM$(&HE120,16)=HEXCHR$("0C E1 C9 78 D9 47 0E 01 21 FE 3F ED 78 03 D9 ED")
300 MEM$(&HE130,11)=HEXCHR$("79 03 D9 2B 7C B5 C2 2B E1 D9 C9")
310 TIME=0:D=USR1(80):PRINT"NOT BY DMA=":TIME

```

リスト 8-17 DMA を使わずにグラフィックススクロール

```

                                .PHASE 0E100H
                                .Z80
                                ;GRAPHIC SHIFT TO LEFT
                                ;
E100    FE 02                    CP      2          ;CHECK VAR.TYPE
E102    28 04                    JR      Z,OKGO     ;HL POINTS INT.?
E104    3E 0D                    LD      A,13       ;TYPE MISMATCH
E106    DD E9                    JP      (IX)       ;ERROR ROUTINE
                                ;
E108    7E                      OKGO: LD      A,(HL)  ;0-255
E109    B7                      OR      A
E10A    C8                      RET      Z          ;0 TIME
E10B    57                      LD      D,A        ;COPY TIMES
                                ;
E10C    01 4000                 LOOP0: LD      BC,4000H ;BLUE
E10F    CD E123                 CALL    SHIFT1
                                ;
E112    01 8000                 LD      BC,8000H ;RED
E115    CD E123                 CALL    SHIFT1
                                ;
E118    01 C000                 LD      BC,0C000H ;GREEN
E11B    CD E123                 CALL    SHIFT1
                                ;
E11E    15                      DEC     D
E11F    C2 E10C                 JP      NZ,LOOP0
                                ;
E122    C9                      RET              ;RET TO BASIC
                                ;
                                ;SUBROUTINE BC=TOP OF PAGE
E123    78                      SHIFT1: LD      A,B
E124    D9                      EXX
E125    47                      LD      B,A
E126    0E 01                   LD      C,01    ;B'C'=SOURCE
E128    21 3FFF                 LD      HL,3FFFH ;COUNTER
                                ;
E12B    ED 78                 LOOP1: IN      A,(C) ;GET 1 BYTE
E12D    03                     INC     BC    ;INC POINTER
E12E    D9                     EXX

```



E12F	ED 79	OUT	(C),A	;COPY 1 BYTE
E131	03	INC	BC	;INC POINTER
E132	D9	EXX		
E133	2B	DEC	HL	;DEC COUNTER
E134	7C	LD	A,H	
E135	B5	OR	L	;CHECK COUNTER
E136	C2 E12B	JP	NZ,LOOP1	
;				
E139	D9	EXX		
E13A	C9	RET		;RET TO MAIN ROUTINE
;				
		END		

#### 1) リスト 8-18 (上1ドットグラフィックススクロール)

律儀に上下をつなげてみた。200回スクロールすると元に戻るわけである。メインメモリの E100<sub>H</sub>番地から E8CF<sub>H</sub>までの 07CF<sub>H</sub>+1バイトをワークエリア(バッファ)にしていることに注意。このサンプルプログラムのように上下をつなげる場合にはどうしても必要である。本当のところは、1ラインのスクロールならワークエリアは80バイトで済むのだが、そうすると手間がかかり、速度に響いてしまう。

79<sub>H</sub>はポート B →ポート A への転送を指定している。

#### リスト 8-18 上1ドットグラフィックススクロール

```

100 DEFINT A-Z
110 CLEAR&HFFFF
120 DEFUSR0=&HE000
130 MEM$(&HE000,12)=HEXCHR$("78 EB 01 80 1F 04 ED A3 3D 20 FA C9")
140 INIT:KLIST0:CLS4:CIRCLE(320,100),60,6
150 PAINT(320,100),&H62
160 SYMBOL(282,100),"朝です",2,2,7,0,PRESET
170 '
180 RESET$=HEXCHR$("C3 C3 C3 C3 C3 C3"):'リセット データ
190 '青画面
200 DMA$=RESET$+HEXCHR$("7D 00 40 CF 07 1C 10 CD 00 E1 9A CF 87"):'TO MEM
210 DMA$=DMA$ +HEXCHR$("7D 00 48 CF 37 1C 18 CD 00 40 9A CF 87"):'IN VRAM
220 DMA$=DMA$ +HEXCHR$("79 00 78 7F 07 1C 10 CD 50 E1 9A CF 87"):'FM MEM
230 DMA$=DMA$ +HEXCHR$("79 80 7F 4F 00 1C 10 CD 00 E1 9A CF 87"):'FM MEM
240 '赤画面
250 DMA$=DMA$+RESET$
260 DMA$=DMA$ +HEXCHR$("7D 00 80 CF 07 1C 10 CD 00 E1 9A CF 87"):'TO MEM
270 DMA$=DMA$ +HEXCHR$("7D 00 88 CF 37 1C 18 CD 00 80 9A CF 87"):'IN VRAM
280 DMA$=DMA$ +HEXCHR$("79 00 B8 7F 07 1C 10 CD 50 E1 9A CF 87"):'FM MEM
290 DMA$=DMA$ +HEXCHR$("79 80 BF 4F 00 1C 10 CD 00 E1 9A CF 87"):'FM MEM
300 '緑画面
310 DMA$=DMA$+RESET$
320 DMA$=DMA$ +HEXCHR$("7D 00 C0 CF 07 1C 10 CD 00 E1 9A CF 87"):'TO MEM
330 DMA$=DMA$ +HEXCHR$("7D 00 C8 CF 37 1C 18 CD 00 C0 9A CF 87"):'IN VRAM
340 DMA$=DMA$ +HEXCHR$("79 00 F8 7F 07 1C 10 CD 50 E1 9A CF 87"):'FM MEM
350 DMA$=DMA$ +HEXCHR$("79 80 FF 4F 00 1C 10 CD 00 E1 9A CF 87"):'FM MEM
360 FORK=1TO200:D$=USR0(DMA$):NEXT
370 END

```

#### 2) リスト 8-19 (上2ドットグラフィックススクロール)

これも律儀に上下をつなげてある。残念ながら実行してよく見ていると色がズレまくっている。ここらへんがDMAでグラフィックを扱う限界なのかもしれないが、「正しい手抜き法」を使えばある程度の改善はできるはずである。挑戦してみていただきたい。

#### リスト 8-19 上2ドットグラフィックススクロール

```

100 DEFINT A-Z
110 CLEAR&HFFFF
120 DEFUSR0=&HD000
130 MEM$(&HD000,12)=HEXCHR$("78 EB 01 80 1F 04 ED A3 3D 20 FA C9")

```



```

140 INIT:KLIST0:CLS4:CIRCLE(320,100),60,2
150 PAINT(320,100),2,2
160 SYMBOL(240,100),"赤丸急上昇",2,2,7,0,PSET
170 '
180 RESET$=HEXCHR$("C3 C3 C3 C3 C3 C3"):リセット データ
190 '青画面
200 DMA$=RESET$+HEXCHR$("7D 00 40 CF 07 1C 10 CD 00 E0 9A CF 87"):TO MEM
210 DMA$=DMA$+HEXCHR$("7D 00 48 CF 07 CD 00 E8 CF 87"):TO MEM
220 DMA$=DMA$+HEXCHR$("7D 00 50 FF 2F 18 CD 00 40 CF 87"):IN VRAM
230 DMA$=DMA$+HEXCHR$("79 00 70 7F 07 10 CD 50 E0 CF 87"):FM MEM
240 DMA$=DMA$+HEXCHR$("79 00 78 7F 07 CD 50 E8 CF 87"):FM MEM
250 DMA$=DMA$+HEXCHR$("79 80 77 4F 00 CD 00 E0 CF 87"):FM MEM
260 DMA$=DMA$+HEXCHR$("79 80 7F 4F 00 CD 00 E8 CF 87"):FM MEM
270 '赤画面
280 DMA$=DMA$+RESET$
290 DMA$=DMA$+HEXCHR$("7D 00 80 CF 07 1C 10 CD 00 E0 9A CF 87"):TO MEM
300 DMA$=DMA$+HEXCHR$("7D 00 88 CF 07 CD 00 E8 CF 87"):TO MEM
310 DMA$=DMA$+HEXCHR$("7D 00 90 FF 2F 18 CD 00 80 CF 87"):IN VRAM
320 DMA$=DMA$+HEXCHR$("79 00 B0 7F 07 10 CD 50 E0 CF 87"):FM MEM
330 DMA$=DMA$+HEXCHR$("79 00 B8 7F 07 CD 50 E8 CF 87"):FM MEM
340 DMA$=DMA$+HEXCHR$("79 80 B7 4F 00 CD 00 E0 CF 87"):FM MEM
350 DMA$=DMA$+HEXCHR$("79 80 BF 4F 00 CD 00 E8 CF 87"):FM MEM
360 '緑画面
370 DMA$=DMA$+RESET$
380 DMA$=DMA$+HEXCHR$("7D 00 C0 CF 07 1C 10 CD 00 E0 9A CF 87"):TO MEM
390 DMA$=DMA$+HEXCHR$("7D 00 C8 CF 07 CD 00 E8 CF 87"):TO MEM
400 DMA$=DMA$+HEXCHR$("7D 00 D0 FF 2F 18 CD 00 C0 CF 87"):IN VRAM
410 DMA$=DMA$+HEXCHR$("79 00 F0 7F 07 10 CD 50 E0 CF 87"):FM MEM
420 DMA$=DMA$+HEXCHR$("79 00 F8 7F 07 CD 50 E8 CF 87"):FM MEM
430 DMA$=DMA$+HEXCHR$("79 80 F7 4F 00 CD 00 E0 CF 87"):FM MEM
440 DMA$=DMA$+HEXCHR$("79 80 FF 4F 00 CD 00 E8 CF 87"):FM MEM
450 FORK=1TO100:D$=USR0(DMA$):NEXT
460 END

```

以上が転送であった。DMAの動作には、転送以外にサーチ、サーチ+転送がある。これらの使い方も基本的には同じなのだが、多少趣が異なるので、まずはごくごく基本のサンプルを示す。

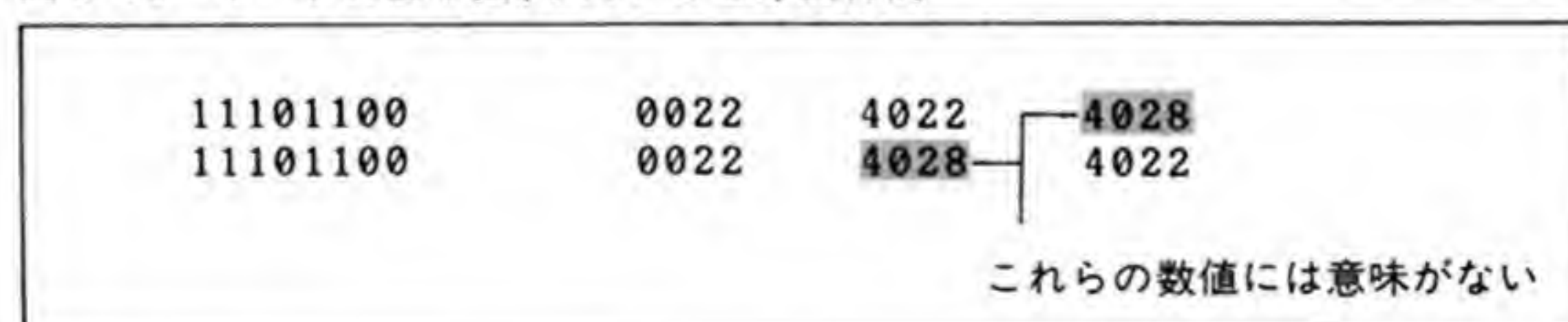
#### K) リスト 8-20

図 8-10 は実行結果である。まず 1000 行からのサブルーチンを説明する。

リードレジスタについては一応説明してあったが、これは DMA が実行を終了した後で、「どれどれお前さんは何をしたのかね」と言いつつノゾキ見るために付いているのである。特にサーチは実行結果が DMA の内部にしか残らないから、どうしてもここを見る必要がある。そのリードレジスタを DMA から読み出すには何とおりの方法があるが、別に深く悩まず全部読み出してしまえばよいのであった。まずリスト 8-20 の 1080 行目からリードレジスタを読み出し、配列 DMA (～) に代入している。

最初に &HBB を OUT しているが、これは WR6 のコマンドで「リード・マスク・フォローズ」(後ろに続くマスクを読め) である。その直後の &H7 がそのマスクである。このマスクの中の対応するビットが立っていれば(1 ならば)、それに対応する 1 バイトのリードレジスタを読み出すことになる。1110 行の FOR～NEXT 文がそうである。図 8-8 を見れ

図 8-10 サーチの基本技(リスト 8-20)実行例





リスト 8-20 サーチの基本技

```

100 INIT:CLS4
110 CLEAR &HFFFF
120 OUT&H4020,&HFF:'マ-ク
130 RESET$=HEXCHR$("C3 C3 C3 C3 C3 C3"): 'リセット デ-タ
140 '
150 'ポート A を使って、マスクなしで&B11111111=&HFFをサーチ
160 DMA$=RESET$+HEXCHR$("7E 00 40 30 00 1C 9C 00 FF C1 9A CF 8B 87")
170 GOSUB"DISPDMA"
180 '
190 'ポート B を使って、マスクなしで&B11111111=&HFFをサーチ
200 DMA$=RESET$+HEXCHR$("62 30 00 18 9C 00 FF CD 00 40 9A CF 8B 87")
210 GOSUB"DISPDMA"
220 END
230 '
1000 LABEL"DISPDMA"
1010 GOSUB"SETDMA":GOSUB"READDMA":GOSUB"DMPDMA"
1020 RETURN
1030 '
1040 LABEL"SETDMA"
1050 FORI=1TOLEN(DMA$):OUT&H1F80,ASC(MID$(DMA$,I,1)):NEXT
1060 RETURN
1070 '
1080 LABEL"READDMA"
1090 OUT&H1F80,&HBB
1100 OUT&H1F80,&H7F
1110 FORI=1TO7:DMA(I)=INP(&H1F80):NEXT
1120 RETURN
1130 '
1140 LABEL"DMPDMA"
1150 PRINT RIGHT$("00000000"+BIN$(DMA(1)),8),
1160 FORI=2TO6STEP2:GOSUB"DMPDMA-SUB":NEXT
1170 PRINT
1180 RETURN
1190 '
1200 LABEL"DMPDMA-SUB"
1210 PRINTRIGHT$("000"+HEX$(DMA(I)+DMA(I+1)*256),4),
1220 RETURN

```

ば分かるように、最初の 1 バイト目はステータスバイトと呼ばれ、各ビットごとに意味を持つ。その後の 2～7 バイト目はそれぞれバイトカウンタ、ポート A アドレス、ポート B アドレスの上下バイトになっている。ただしこれらの値は DMA の構造上から多少大きめの値を示すようになっているので注意が必要である。図 8-10 を見ていただきたい。たしかに、ポート A のアドレスが &H4022 となっている。

それでは 160 行の HEXCHR\$ の中のコマンドについて解説する。まず、サーチを実行するためには開始アドレスを決め、ポート A、B のどちらかに割り当てなければいけない。A、B どちらでもよいのだが、その場合は WR0 でソース側（転送元）に指定しておく必要がある。リスト 8-20 ではポート A（160 行）ポート B（200 行）の両方を使った場合を示してある。ポートを取り替えたこと以外は同じことをさせている。さて、中程に 9C<sub>H</sub> というのがあるが、これが WR3 である。その直後の 00 と FF が WR3 のサブレジスタへの値である。図 8-5 を参照。マスクバイト（00）というのは「ビットが 0 なら比較しろ」という意味を持つ。マッチバイトはサーチすべき 1 バイトのデータである。マスクバイトというのが出てくると、とたんにこんがらがってしまうのだが、

（マッチバイト）OR（マスクバイト）＝（ポート A）の指すデータ）OR（マスクバイト）となれば一致成立ということであると覚えておけばよいだろう。9C<sub>H</sub> は D<sub>2</sub> が 1 であるからストップ・オン・マッチを指定している。さて、他のコマンドは 8B<sub>H</sub> 以外は簡単に分かるだろう。8B<sub>H</sub> は WR6 のステータスバイト初期再設定コマンドである。理由は不明なのだ



が、サーチを行なう場合はロードコマンド(CF<sub>H</sub>)の後にこのコマンドを送ってやる必要があるみたいである。手元にある資料にはサーチの実例がほとんどなく、また説明も不十分なのである。とにかく、8B<sub>H</sub>を取り除くと動作がおかしくなる。

#### L) リスト 8-21

この中では、ポート A を使ってサーチしている。最初の 170 行の例は I/O 空間(グラフィック RAM)中の 4021<sub>H</sub>番地に OUT してある FF<sub>H</sub>を、4000<sub>H</sub>からサーチしている。200 行は同じように F0<sub>H</sub>をサーチしている。このデータはないので、ブロックレングス (2F<sub>H</sub>) + 1 個のデータをサーチした後に止まる。転送と同じように、ブロックレングスで指定した値 + 1 が対象になるのである。230 行は FC<sub>H</sub>をサーチしている。これはちょうどサーチ範囲の最後にあるので、ステータスバイト(図 8-11 の 3 行目)の D<sub>5</sub> (= 0 でエンド・オブ・ブロック), D<sub>4</sub> (= 0 でデータが一致)の両方とも 0 になっている。260 行はマスクを使ってサーチしている。

$$(FF_H) \text{ OR } (0F_H) = FF_H$$

$$(4020 \text{ の内容} = FE_H) \text{ OR } (0F_H) = FF_H$$

であるから、これは 4020<sub>H</sub>の FE<sub>H</sub>で一致している。290 行は、マスクバイトが FF<sub>H</sub>であるから、いかなるデータとでも一致することになる。なぜなら、

$$(\times \times) \text{ OR } (FF_H) = FF_H$$

$$(\triangle \triangle) \text{ OR } (FF_H) = FF_H$$

だからである。

320 行はサーチはサーチなのだが、WR3 へのコマンド中で「見つけたら停止」を指定していない(98<sub>H</sub>)ので、指定したバイト数だけ最後まで処理している。ただしステータスバイトの D<sub>4</sub> はちゃんと 0 (発見した)になっている。

次にとても面倒なのだが、読み出したレジスタの値は動作モード、条件などによってかなり複雑に変化する。正しくは参考文献 4 を見ていただきたいが普通に使う分には「バーストモードのサーチで、データが一致したなら、ソースポートのアドレスは発見アドレスの + 2」と覚えておけばよいだろう。その他の場合は実に変幻自在で、このような状況を押付けられると、どうも使う気が起きなくなってしまうものである。ただでさえ転送と違って、利用法の少ない動作なのだからなおさらなのである。さらには、もし WR3 に「一致不成立時に停止」という動作が指定できたなら、「0 以外のデータを見つける」ということもできたはずなのである。実になまぬるい。

図 8-11 サーチ六態(リスト 8-21)実行例

11101100	0023	4023	4028
00011000	0030	4030	4028
00001000	0030	4030	4028
00101000	0022	4022	4028
11101100	0002	4002	4028
00001000	0030	4030	4028

これらの数値には意味がない



リスト 8-21 サーチ六態

```

100 INIT:CLS4
110 CLEAR &HDFFF
120 OUT&H4020,&HFE
130 OUT&H4021,&HFF
140 OUT&H402F,&HFC
150 RESET$=HEXCHR$("C3 C3 C3 C3 C3 C3"): 'リセット データ
160 'マスクなしで&B11111111=&HFFをサーチ
170 DMA$=RESET$+HEXCHR$("7E 00 40 2F 00 1C 9C 00 FF C1 9A CF 8B 87")
180 GOSUB"DISPDMA"
190 'マスクなしで&B11110000をサーチ
200 DMA$=RESET$+HEXCHR$("7E 00 40 2F 00 1C 9C 00 F0 C1 9A CF 8B 87")
210 GOSUB"DISPDMA"
220 'マスクなしで&B11111100をサーチ
230 DMA$=RESET$+HEXCHR$("7E 00 40 2F 00 1C 9C 00 FC C1 9A CF 8B 87")
240 GOSUB"DISPDMA"
250 'マスクありで&B1111????をサーチ
260 DMA$=RESET$+HEXCHR$("7E 00 40 2F 00 1C 9C 0F FF C1 9A CF 8B 87")
270 GOSUB"DISPDMA"
280 'マスクありで&B????????をサーチ
290 DMA$=RESET$+HEXCHR$("7E 00 40 2F 00 1C 9C FF FF C1 9A CF 8B 87")
300 GOSUB"DISPDMA"
310 'マスクなしで&B11111111をサーチただし、停止しない
320 DMA$=RESET$+HEXCHR$("7E 00 40 2F 00 1C 98 00 FF C1 9A CF 8B 87")
330 GOSUB"DISPDMA"
340 END
350 '
360 'このあとには、リスト8-20の1000行目以後を入力してください。

```

まあ、文句はこれくらいにして、次には結構見えそうなモードのサーチ+転送のサンプルを示すのである。

M) リスト 8-22

サーチ+転送の使い方は、WR0で「転送/サーチ」を指定することと、WR3によりマスクとマッチバイトを指定する以外は(ロードの後の8B<sub>H</sub>もあるけれど)転送と基本的に同じである。もちろんWR3で「見つけたら停止」を指定しなければ、転送したデータの中に一致するデータがあったということだけが後から分かる。サンプルプログラムの中では「.」(ピリオド)までの文字列をテキスト VRAM へ転送している。もちろんこのとき、ブロック長は充分大きな値にしておかなければ、「.」に達する前に動作が停止してしまう。このサンプルを作ってみて、またもや不満が出てしまった。すなわち、「一致する一つ前のデータまで転送」というモードが欲しいのである。

ま、文句ばかり言っても仕方ないか。

リスト 8-22 サーチ+転送

```

100 INIT:CLS4
110 CLEAR &HDFFF
120 MESSAGE$="WATASI WA DMA DE TENSOU SARERU."
130 MEM$(&HE000,LEN(MESSAGE$))=MESSAGE$: 'ソースを用意する
140 LOCATE0,0:PRINTSPACE$(80): 'アトリビュートの設定
150 RESET$=HEXCHR$("C3 C3 C3 C3 C3 C3"): 'リセット データ
160 'マスクなしで、"."までを、サーチ+転送
170 DMA$=RESET$+HEXCHR$("7F 00 E0 00 01 14 18 9C 00 2E CD 00 30 9A CF 8B 87")
180 LOCATE0,3:GOSUB"DISPDMA"
190 END
200 '
210 'このあとには、リスト8-20の1000行目以後を入力してください。

```



以上で DMA の説明は終わる（ほんとは第 9 章のディスクのところでもたやるけど）。で、DMA の良い点の一つとして挙げることができるのが、BASIC 上から機械語を使わずに、機械語と同じ速度、もしくはそれ以上の速度で処理ができるということである。さんざんけなしたけど、サーチだって BASIC でやったのなら実にトロいことになってしまうのだ。とりあえず DMA にありがとうである。



第

9

章

# フロッピーディスク



ディスクを回すのである



## ■第9章

# ディスクを回すのである……………

この章はフロッピーディスクである。そこでまずは参考書について一言。

フロッピーディスク関係の本の中で、ストロングタイプのお勧め品は、参考文献 10 にも挙げてある『最新フロッピー・ディスク装置とその応用ノウハウ』である。この本は 1984 年の発刊で、残念ながらあんまり「最新」ではなくなってしまったのだが、プロテクト以外のことに限っては、ほとんどすべてが載っているので、是非とも手元に置いておきたい一冊である。

さて、同書によるとフロッピーディスクなるものがこの世に生を受けたのは、1972 年のことである。IBM の大型コンピュータに採用されたのだが、確か私の記憶だと、IPL 起動時のシステム読み込みに使われていたはずである。このときのものは、8 インチの片面単密で、容量は約 250K バイトであった。当時としては、そのように簡単に取り外しができて、しかもランダムアクセス可能な大容量のデバイスは画期的なものだったらしい。

時は移り、それから 4 年後の 1976 年に米国シュガート社から 5.25 インチディスクが発表された(なんでキッチリ 5 インチにしなかったんだっ。無意味な不満とともに、以下ではしばしば 5 インチと呼ぶ)。これは当初は容量が 80K バイトしかなかったそうである。

8 インチと 5 インチは、その後着々と世に広がり、また技術の向上に伴って記録密度も上がってくるのだが、三つ子の魂百までと言うように、8 インチと 5 インチはちょっと違う道を歩むことになる。すなわち、巨人 IBM の手によって生み出された 8 インチの方は「IBM フォーマット」という、誰も逆らえないスタンダードが根底に確立したのだが、片や 5 インチの方は、ほとんど野放しに近い状態となり、いろいろなフォーマットや方式が乱立することになった。

その中でも一番の被害者は 5 インチドライブを持つ CP/M ユーザーである。8 インチドライブの場合は「片面単密度 IBM フォーマット」と一声念じれば、ディレクトリなどに相当するソフト的なフォーマットも含めてすべてが解決したのだが、5 インチの場合は、単密度しか読めないやつとか、倍密度しか読めないやつとか、内部のディレクトリのエンタリーの数(登録できるファイルの数)が違ったりとか、トラックの数が違ったりとかで、も一むちゃくちゃになってしまったのだ。その状況は今も変わらず、メディアの互換性はほとんど捨ておかれてしまっている。その状況は 3.5 インチでも同じである。ただ一つ明るい材料といえば、5 インチでは MS-DOS フォーマットが実質的な標準になりつつあるということだが、それとても MS-DOS フォーマットへのコンバータが必ず入手できるわけではない。困ったもんだ。

なおこの章のサンプルプログラムは、特にことわっていない限り 2D モードで動くようになっているので turboZ のユーザーの方などは注意されたい。



## フロッピーディスクドライブの種類

今まで「5 インチ」、「8 インチ」、「2D」、「2DD」、「2HD」と勝手に書いてきたが、これからちゃんと説明しておくことにする。

太古の昔には、さまざまな形式のものがあったそうだが、現在の主流はメディアのサイズが、

**8, 5.25, 3.5, 3 インチ**

記録方式が、

**1S, 2S, 1D, 2D, 2DD, 2HD**

となっている。表 9-1, 9-2 が大まかな分類と大体の容量、個人的な予測である。

表 9-1 メディアの種類

	5.25, 3.5, 3 インチ		8 インチ	
	片面	両面	片面	両面
単密度	1S (80KB)	2S (160KB)	1S (250KB)	2S (500KB)
倍密度	1D (160KB) 1DD (320KB)	2D (320KB) 2DD (640KB) 2HD (1MB)		2D (1MB)

表 9-2 普及度の(個人的な)予測

サイズ	メジャー度
8	↓
5.25	↗
3.5	↗
3	+

表を見て当然気付くだろうが、記録方式では 8 インチの方に 1D がない(噂によると、どこぞに 1D の 8 インチドライブがあるそうだが、多分 2D のドライブの片面だけを使っているのではないかと推測する)。また、2DD や 2HD に相当するものもない。理由は簡単である。「IBM が作らなかった」からである。それに対して、5, 3.5, 3 インチの方は 1D と 2S, 1DD と 2D のように、同じ容量なのに違う形式のものがあったりする。この理由も明白である。「IBM が作らなかった」からである。

まずはディスクの回り方などについて説明する。

ディスクの回転速度は 8 インチが毎分 360 回転、5 インチでは 2HD が 8 インチと同じく毎分 360 回転、それ以外は 300 回転である。3.5 インチは 300 回転と 600 回転の 2 種類があるが、メディアに記録される信号は同じで互換性はある。3 インチは毎分 300 回転で、



5インチと同じ。回る方向は、5インチで確認したところ、ラベル側から見て右回り（時計回り）であった。8インチはよく見えなかったので不明だが、多分同じであろう。

次にディスクの形状であるが、まずはライトプロテクトの方式について少し触れておく。8インチの場合は、ディスクの一部を専用の器具を使って切り欠くことによって、ライトプロテクトの状態になる。切り欠いた所に銀紙を張ると元どおりに書き込めるようになる。5インチの場合は逆で、元々切り欠きがあり（最近のゲームソフトなどでは切り欠きのない特別のディスクセットを使っているものもあるが）、その状態が書き込み可能である。そこへ銀紙を張るとライトプロテクトされる。だから同じ銀紙でも、8インチの場合と5インチの場合は機能が逆である。

3.5インチの場合は、プラスチック製のスイッチがあり、それをスライドさせることによりライトプロテクトになる。スイッチだから元に戻すのも簡単である。

3インチも3.5インチと同様にプラスチックのスイッチをスライドさせる方式である。ただしスイッチは二つある。この理由は後程述べる。

なお、ライトプロテクトであるが、X1の場合はハード的に書き込みができないようになるが、一部の機種では絶対に書き込めないとはいえず、コントロールプログラムの方でライトプロテクトになっているかどうかをチェックしてやって、なっていたら書き込み動作を中止してやる、という処理が必要なものもある。

次にインデックスホールであるが、8、5、3インチにはあるが、3.5インチにはない。代わりに、3.5インチのディスクには位置決めホールがあり、それがインデックスホールと同じ役目を果たす。

実は3インチにも位置決めホールに相当する部分があり、別にインデックスホールがなくてもよかったのではないかという気がするが、しっかりとある。

次に「裏表」についてである。3.5、3インチでは不明だが、8、5インチではラベルを張る面がサイド1、反対側がサイド0である。1S、1Dなどの片面しか使わないドライブではサイド0だけを使っている（ただし例外もある）。

このサイドに関しては、3インチが変わっていて、1D用のドライブに対しては、(2D用の)ディスクを裏返して使うことができる。これがライトプロテクト用のスイッチが二つもある理由である。ただしX1の3インチドライブは2D用だから、そのようなことは関係なく、ディスクを裏返してセットすることはできない。

次に記録形式についてである。

まずは1S、2Dなどの形式について説明を行なっておく。最初に付いている数字は、

「1」が片面

「2」が両面

を意味している。

次にはアルファベットが1もしくは2文字付くが、これは、

「S」=単密度(Single density)

「D」=倍密度(Double density)

「DD」=倍トラック倍密度(Double track Double density)



## 「HD」=高密度(High Density)

の意味である。

まず、片面と両面の差であるが、これはそのものずばりであるから理解しやすいはずである。片面ドライブと両面ドライブの差は、要するにリード/ライト用のヘッドが一つだけか、それとも向かい合って二つ付いているかである。ただし両面ドライブの二つのヘッドはぴったりと向かい合っているわけではなく、少し前後に（数トラック分）ずれている。

次に「S」、「D」などについてである。これはまず、「S」と「その他」に分けられる。「S=単密度」は別名「FM 記録方式」であり、「S」以外はすべて「MFM 記録方式」である。FM は「Frequency Modulation」（周波数変調方式）の略で、これはラジオの FM と同じである。つまり FM とはラジオだけのことではないのである。これに対して、MFM は「Modified Frequency Modulation」である。Modified は、「修正された」というような意味である。詳しいことは他の参考書に譲るが、FM 記録方式ではクロックパルスというパルスが一定時間（5 インチ 1S, 2S などの場合で  $8\mu\text{s}=0.000008\text{秒}$ ）ごとに書かれ、その間にデータ用のパルスがあれば「1」、なければ「0」である（データは 1 ビットだけ）。なぜこれが FM なのかといえ、全部「1」を書いた場合はパルスの間隔が  $4\mu\text{s}$  になり、それに対して全部「0」を書いた場合はパルスの間隔は  $8\mu\text{s}$  である。つまりパルスの来る周期が  $4\mu\text{s}$  か  $8\mu\text{s}$  かで 1, 0 を区別しているのだ。だからやっぱり周波数変調=FM である。

MFM はどうなのかというと、この方式は大胆にも、「前のデータが 0 で、なおかつ後ろのデータも 0 ならば、間にクロックパルスを入れる。それ以外の場合にはクロックパルスは書かない」ということになっている。このように修正してやると、「もっとも間隔の狭いパルスの間」は FM 方式のときの 2 倍になる。となれば、ディスクの磁性体とヘッドの性能の余裕は 2 倍になる（識別できる二つのパルスの間隔が「性能」である）。そこでほっと一息ついた磁性体とヘッドには気の毒だが、クロックパルスの間隔も入れた書き込み速度全体を 2 倍にしてやる。これで磁性体やヘッドの性能は基本的に同じままなのに（本当はそうは単純にいかないのだが）、倍のデータを読み書きできることになる。これが倍密度=MFM である。早い話が、クロックビットを適当に間引くということなのである。「それじゃ、ディスクの回転と同期を取れなくなるじゃないか」という疑問が出てくるであろう（私もそう思う）。しかしディスクに書かれているのはデータだけではなく、あちこちに同期を取るための特殊なパルスも書かれていて、ちゃんと読み書きができるのである。これは後の「物理フォーマット」の所で説明する。

次に「DD」、「HD」についての説明である。「DD=倍トラック倍密度」であるが、当然のことながら、ディスクにはトラックというものがあることは知っているだろう。これはバームクーヘンの年輪みたいなものである。このトラックに沿ってデータが書かれるのだが、倍トラックというのはこのトラックの間隔を半分にして、数を倍にしたものである。結局、めでたく容量は 2 倍になる。

最後が「HD」だが、これは 8 インチと同じ形式を 5 インチに持ってきたものである。トラック数は 2D と 2DD の中間であるが、パルス間隔が  $2\mu\text{s}$  と、半分になり、1 トラック当



たりのデータが多くなっている。それにより全体の容量は2DDの約1.5倍となる。

以上、あれこれ書いたが、結論として言えるのは、8インチ以外のディスクは混沌に満ちているということである。おそらくこれは、急速に技術が向上しているものすべてに言えることだろうが、この先どうなるのか考えてみただけでも恐ろしい限りである。最近では2インチとかも出てきたし。

## トラック、セクタなどなど

まず用語について説明する。

### ① サイド(side)

表と裏の面のことである。サイド0, 1がある。

### ② トラック(track)

1周分のデータを指す。

2Dは1サイド当たり40トラック、2DDは1サイド当たり80トラック、2HDは1サイド当たり77トラックである。なお、サイド0, 1をまとめて数えて、「2Dは80トラック、2DDは160トラック」などと言う場合があるが、これは「誤解してください」と言っているのと同じである。できれば、「2Dは一つのサイド当たり40トラックで、サイドは二つある」と表現した方がよい(後述のシリンダを使う手もある)。なお、トラックの番号は外側から0番, 1番……と数える。

ちなみに、サイド0と1では、実際の位置は各トラックで4本分(2DDでは8本)ずれている。サイド0の方が外側である。これはヘッドの位置がそもそも4本分(8本分)ずれているためである。このずれは、ディスクを使ううえではまったく意識する必要はない。

各ディスクにおけるトラック間距離は一定で、表9-3に示すとおりである。このことから、FDDというものが高い精度を必要とするデリケートなものであることが分かるだろう。

表9-3 各メディアにおけるトラック間距離

サイズ	トラック間隔(mm)
8インチ	0.529
5インチ(2D) (2DD, 2HD)	0.529 0.2646
3.5インチ(2D) (2DD, 2HD)	0.375 0.1875
3インチ	0.254

### ③ シリンダ(cylinder)

サイド0, 1の同じ番号のトラックを併せて1シリンダと呼ぶ。2Dの場合は1枚のディスクに40シリンダあることになる。シリンダもトラックと同様に外側から0番, 1番……と数える。



#### ④ セクタ(sector)

1トラックを何個かに分解してセクタと呼ぶ。実際にデータを読み書きするのはこの部分である。5インチの2D、2DDでは普通1トラック(片面)当たり16セクタである。セクタはトラックと違い、1番、2番、……と番号が付けられている。なお、セクタの数は16個とは限らない。5インチの2Dのまともなフォーマットでは、1セクタの容量を256バイト、512バイト、1024バイトの中から選べる。それぞれの場合において、1トラック中のセクタの数は16、9、5となり、ディスク1枚当たりの容量は320Kバイト、360Kバイト、400Kバイトとなる。ただし裏技で、1トラックに17セクタを作ったりすることも可能である。

8インチ2Dと2HDの場合、セクタの容量と、1トラック(片面)当たりのセクタ数は、256バイト→26セクタ、512バイト→15セクタ、1024バイト→8セクタとなる。

当然ながら「容量は大きい方がよいのだから、1セクタは1024バイトにして、1枚当たりの容量を400Kバイトにすべきだ」と考えるかもしれないが、そうした場合、良いことばかりとは限らない。第一に読み書きは1セクタごとに行なわなければならないので、バッファは1Kバイト必要になる。さらには、1セクタを読み書きする時間も長くなる。結局ディスクの容量は2割増しになるという利点以外は、「1セクタ256バイトのフォーマットで、常に4セクタをまとめて読み書きしている」と似たようなことになってしまうのである。さらには、他機種や、DOSとのデータ互換性なども問題になってくる。

## 本題に入る前に

本当はディスクエディタを載せるべきなのかもしれないが、敢えて載せないことにした。読み出して表示するだけであればBASICに付属の“Device dump”があるが、書き込みの際には別途にツールが必要になる。

## では、始める

まずはX1 HuBASICのソフト的なフォーマットについて書く。

一般的に、フロッピーディスクのフォーマットというと2とおりある。「物理フォーマット」と「ソフトフォーマット」である。物理フォーマットというのは、ディスクett上に作られた区画のようなものである。前述したように、1セクタの容量などが物理フォーマットによって決められる。BASICに付属している“FORMAT & COPY. Uty”が物理フォーマットをディスクettに書き込むわけである。物理フォーマットは1トラックごとに設定しなければならない点が、普通のデータ書き込みと異なる。普通、5インチの2D、2DDでは、1トラック分のフォーマットを設定するには、6.25Kバイトのデータを必要とする。

ソフトフォーマットというのは、「ファイルの格納場所」や「IPL起動するファイルは、このような形式でディスクettに書かれていなければならない」などを定めるもので



ある。

X1 のソフト的なフォーマットの要点は大体次の 3 点である。

- ① ディレクトリの構造
- ② FAT(File Allocation Table)の構造
- ③ IPL 起動プログラムの指定

①のディレクトリというのは、要するに「このディスクにはこんな名前のファイルが入っている」ということなどを記録してある部分である。ファイル名以外にも、ファイルの大きさや、書き込まれた日付などもある。つまり名簿のようなものである。②の FAT は、現在ディスクのどの部分がどのように使われていて、どの部分が空いているかを示す部分である。③の IPL のプログラム指定は、ディレクトリ的一种だと思えばよい。ではまず、ディレクトリの説明から始める。

ディレクトリの構造は表 9-4 (turbo の USER'S MANUAL より一部変更) である。一つのディレクトリは  $32=20_{\text{H}}$  バイトからなる。中身についての前に、レコード番号とクラスタ番号を説明しておく。レコード番号は 5 インチ 2D と 3 インチ 2D では全部で  $16 \times 2 \times 40 = 1280$  個あるセクタに番号を付けたもので、

$$\begin{aligned}\text{レコード番号} &= (\text{セクタ番号} - 1) \\ &\quad + (\text{トラック番号} \times 32) \\ &\quad + (\text{サイド番号} \times 16)\end{aligned}$$

である。つまり、

トラック 0, サイド 0  
トラック 0, サイド 1  
トラック 1, サイド 0  
トラック 1, サイド 1  
⋮  
⋮

と順番を付けて各セクタに番号を振ったものである。

クラスタというのは、16 セクタをひとまとめにしたもので、FILES 命令のときに出てくる「○△ Clusters free」の Cluster である。第 0 クラスタは第 0 ～15 レコード、次の第 1 クラスタは第 16 ～31 レコード……となる。はっきり言って 5 インチ 2D と 3 インチ 2D では、トラック 1 本分 (片面) である。

そこでディレクトリの中身の説明であるが、第 0 バイトはファイルのアトリビュート (属性) を示している。各ビットごとに意味が割り振られているのだが、 $00_{\text{H}}$  と  $\text{FF}_{\text{H}}$  だけは特別で、 $00_{\text{H}}$  は KILL されたファイル、 $\text{FF}_{\text{H}}$  はこれから先はまだ使用されたことのないディレクトリの領域であることを示す。つまり第 0 バイトが  $\text{FF}_{\text{H}}$  であるディレクトリ領域に出合ったら、後は見る必要がないということである。

各ビットの意味は表にあるとおりだが、turbo で階層化ディレクトリが採用されたことにより、それまで X1 で予備になっていた第 7 ビットが割り当てられた点に注意。また、第



表 9-4 ディレクトリの内容

	内 容
0 バイト目	種類を表す。 00 は KILL されたファイルまたは未使用領域。FF は使用ディレクトリテーブルの終わり。 bit 0 が 1 …… Bin ファイル(機械語で書かれたファイル) bit 1 が 1 …… Bas ファイル(BASIC テキストで書かれたファイル) bit 2 が 1 …… Asc ファイル(ASCII セーブされたファイル) bit 4 が 1 …… FILES で表示しない：0 …表示する bit 5 が 1 …… リードアフターライト ON：0 …OFF bit 6 が 1 …… 書き込み禁止ファイル：0 …書き込み OK bit 7 が 1 …… 下位ディレクトリ bit 3 は予備
1 バイト目～13 バイト目	ファイル名(13 文字)
14 バイト目～16 バイト目	ユーザー指定 EXTENTION エリア(3 文字)
17 バイト目	パスワードのバック(無指定なら 20 <sub>H</sub> の値)
18・19 バイト目	ファイルのバイト数(Bas および Obj のみ有効)
20・21 バイト目	ファイルのメインメモリ先頭アドレス(Obj のみ有効)
22・23 バイト目	ファイルのメインメモリ実行アドレス(Obj のみ有効)
24 バイト目～28 バイト目	作成された年、月、曜日、時、分が書き込まれている。 例：'84 年 12 月 01 日土曜日、16 時 36 分 先頭から、 年 年 月 曜 日 日 時 時 分 分 84 C 6 01 16 36
29 バイト目～31 バイト目	ファイル先頭 クラスタ値 29 バイト目 HIGH バイト(2D では 0) 30 バイト目 LOW バイト 31 バイト目 MIDDLE バイト(2D では 0)

4 ビットは SET 命令で “S”，第 5 ビットは “R”，第 6 ビットは “P” の指定に対応する。

次に第 1 バイトからの 13 バイトであるが、これはファイル名である。ファイル名が 13 文字に満たない場合は、残りにスペースを詰めることになっている。ファイル名の中にはコントロールキャラクタを入れることも可能であるが、そうすると FILES での表示が狂ったり、単純には読み出せなくなったりする。昔、それをプロテクトに使ったソフトもあった。次に 3 バイトは拡張子と呼ばれるもので、「.」の後に付ける「Bas」とか「Uty」である。

第 17 バイトはパスワードである。パスワードとは、「SAVE “ファイル名；PASS”」で指定できる。実際には文字列なのだが、計算によって 1 バイトに圧縮されてしまっている。だから、違うパスワードであっても、計算の結果同じ 1 バイトのデータになってしまうこともあり得る。パスワードを指定しなかった場合、この 1 バイトは 20<sub>H</sub>になっている。

第 18～23 バイトまでは特に説明する必要はないだろう。

第 24～28 バイトは日付である。サブ CPU から読み出したデータと同じ形式である。

第 29 バイトからの 3 バイトは、ファイル本体の先頭が書き込まれているクラスタ番号を指定している。3 バイトの数値で、High (上)、Low (下)、Middle (中) と、変則的な順序になっていることに注意。次の FAT の所で説明するが、ファイルの最小単位は 1 クラス



タである。つまり、1行しかない BASIC のプログラムをセーブしても1クラスタ=4K バイトを使ってしまう。

次に FAT であるが、これは各クラスタの状態（使われているかどうか/つながりの順序など）を示している所である。X1 のフォーマットでは 2D では第 14 レコード、2DD では第 14、15 レコード、2HD では第 28、29 レコードにある。

まず、各クラスタの状態は次の三つに分かれる。

- a) 使われていない
- b) 使われていて、自分は最後ではない（別のクラスタに続く）
- c) 使われていて、自分が最後のデータを持っている

第 n 番クラスタの状態は、FAT 中の 2 バイトで表される。実際にどの 2 バイトかは、デバイスの容量によって少々複雑なのだが、128 クラスタ以下 (=512K バイト) の容量のデバイス (2D など) では単純で、「第 n クラスタの状態は、FAT 内の第 n バイト目一つ」で分かるようになっている。そこで、ここではその場合に限定して説明する。

まずはその 1 バイトの値が a), b), c) のそれぞれにおいてどうなっているかという

- a) → 00<sub>H</sub>
- b) → 01<sub>H</sub>~7F<sub>H</sub> (次に続くデータを持つクラスタ番号)
- c) → 80<sub>H</sub>~8F<sub>H</sub> (7F<sub>H</sub>を引いた値が、そのクラスタ内で実際に使われているレコード数)

となっている。つまりチェーン状態を表しているのだ。2D などでは第 0 クラスタと第 1 クラスタ (レコード 0 ~ 32) はディレクトリと、FAT 自身に使われているので、FAT の先頭 2 バイトは常に、

01<sub>H</sub>, 8F<sub>H</sub>,

となっている。すなわち「第 0 クラスタは 1 クラスタ全部が使われていて、第 1 クラスタに続く。第 1 クラスタは 8F<sub>H</sub> - 7F<sub>H</sub> = 16 レコードが実際に使われていて、このクラスタでチェーンは終わり」である。

最後の IPL 起動プログラムの指定である。これは第 0 レコードに書き込まれて「ディレクトリ」に似ているものであるが、次の 3 点で普通のディレクトリと違う。

- ① 先頭のアトリビュートは「Bin」の指定でなければならない
- ② ファイル名の拡張子は「Sys」でなければならない
- ③ 先頭クラスタ番号を指定する所には、代わりにレコード番号を書き込んでおく

では最初にディレクトリの書き換えを行なってみる。INIT 命令ですべてのファイルを消去したディスクに内容の違う BASIC テキストを二つセーブする。そうすると BASIC テキストのサイズによって若干異なるだろうが、ディレクトリの中身は大体図 9-1 のようになるはずである。そこで二つのディレクトリの「先頭クラスタ番号」(1E<sub>H</sub> 番目) と「ファイルの大きさ」(12<sub>H</sub>, 13<sub>H</sub> 番目) を交換して書き込んでやる。これで「ファイル名が交換された」のと同じことになる。

もう少しまともな用途としては、うっかり指定した後、忘れてしまったパスワードを解



図 9-1 ディレクトリの例

#001000=02	54	45	53	54	31	20	20	20	20	20	20	20	20	42	61	'TEST1	Ba
#001010=73	20	44	09	00	00	00	00	85	A5	05	05	01	00	02	00	's D1	※
#001020=02	54	45	53	54	32	20	20	20	20	20	20	20	20	42	61	'TEST2	Ba
#001030=73	20	64	02	00	00	00	00	85	A5	05	05	01	00	03	00	's d	※

これらを交換すると「ファイル名が交換された」ことになる

除したりもできる。そのほか日付の書き換えやファイル名の変更もできる。

次に FAT もいじってみる。ここまでくるとやはり KILL されたファイルの復活が面白そうである。

まず、充分な大きさの BASIC テキストを作って欲しい。わざわざ打ち込まなくても RENUMBER と MERGE を繰り返せば長くすることができる。すなわち、

```
OPTION SCREEN 2:INIT "MEM:"
RENUM 1000:SAVE "MEM:BT1",A
RENUM 2000:SAVE "MEM:BT2",A
:
:
```

とした後で、各 ASCII ファイルを MERGE するなどしていけばよい。

十分に長い BASIC テキストを、INIT 命令ですべてのファイルを消去したディスクにセーブすると、ディレクトリと FAT は大体図 9-2 のようになる。そこでそのファイルを KILL すると、おのおのは図 9-3 のようになる。違っているのはディレクトリの第 0 バイトが 00<sub>H</sub>で「消去されたファイル」を示していることと、対応する FAT がすべて 00<sub>H</sub>に書き

図 9-2 ディレクトリと FAT の例

#001000=02	52	45	56	49	56	41	4C	20	20	20	20	20	20	42	61	'REVIVAL	Ba
#001010=73	20	49	2D	00	00	00	00	85	A5	05	07	21	00	02	00	's I-	※
#000E00=01	8F	03	04	8D	00	00	00	00	00	00	00	00	00	00	00	'※	※
#000E10=00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	'	※
#000E20=00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	'	※

図 9-3 ファイルを KILL した直後のディレクトリと FAT の例

#001000=00	52	45	56	49	56	41	4C	20	20	20	20	20	20	42	61	'REVIVAL	Ba
#001010=73	20	49	2D	00	00	00	00	85	A5	05	07	21	00	02	00	's I-	※
#000E00=01	8F	00	00	00	00	00	00	00	00	00	00	00	00	00	00	'※	※
#000E10=00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	'	※
#000E20=00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	'	※

この部分がかわっている



換えられていることである。復活の第一歩に、まずディレクトリに残っているファイルサイズから、何クラスタ（何レコード）だったのかを計算する。例では1クラスタ=4KB=4096バイトより、2クラスタと14レコードだったということが分かる。よって、第2クラスタは第3クラスタへ、第3クラスタは第4クラスタへ、第4クラスタはそこで終わりで、14レコード分のデータを持っているから $7F_H + 14 = 8D_H$ となる。そのように書き換えていくと、結局図9-2のとおりになるはずである。それをディレクトリとFATの所書き込んでやるとファイルは復活する。

しかし実際のファイルの復活はこのように **チョロイものではない**。何度もSAVE、KILLを繰り返したディスクだと、実際のクラスタのつながりが、 $2 \rightarrow 9 \rightarrow 10 \rightarrow 12$ などのように飛び飛びになっている可能性があるのだ。「そのときはFATの $00_H$ のクラスタを順に拾っていけばいいではないか」と思うかもしれないが、それは完全ではない。なぜなら、そのFATの $00_H$ が復活したいファイルに対応しているとは限らないからである。3日前にKILLしたファイルに対応した $00_H$ かもしれないのである。だから現実にはファイルを復活するためには、「あーでもない、こーでもない」のパズルを繰り返さなければならない可能性が高いのである。なお、FATをいじりそこねると「Bad Record」というエラーが出る。場合によっては**暴走**することもあり得る。心するよーに。

ここでturboの階層化ディレクトリについて説明しておく。はっきり言って構造は簡単そのものである。つまり、MKDIRはもともとのディレクトリと同じサイズ=1クラスタのファイルを作るだけなのである。CHDIRが実行されたなら、「そのディレクトリファイルを、ディレクトリのつもりで使う」だけなのである。すなわち、本質的には第1クラスタ（レコード番号16~32）に固定されていたディレクトリ領域を、任意のクラスタに変更できるようにしただけなのだ（本当はもっと複雑だが「本質」はそれだけのこと）。いろいろ試してみて、ダンプさせれば一目で分かるであろう。

次にIPL起動の指定を実際にやってみることとする。

BASIC (CZ-8FB01) を起動し、例によってINIT命令でファイルを全部消去したディスクを作る。そして、

```
POKE &H012B, &H00, &H10
```

```
POKE &H1053, &H00, &H10
```

を実行する。次に、

```
SAVEM "ディスク番号: MONITOR.Sys", &H0000, &H149F, &H0000
```

でモニタとIOCS部分をディスクにセーブする。モニタを書き換えて縁起が悪いので、BASICを再起動する。次に何らかのツールを使って、ディレクトリの最初の1レコードを第0レコードへコピーする。後は第0レコードの30バイト目を、クラスタ番号( $02_H$ )からレコード番号( $20_H$ )に書き換えればOK。なお、IPL起動のプログラムは連続したレコードに書かれている必要がある。

ここから先ではFDCなるLSIに詰め寄り、抜く手も見せずにフロッピーディスクの深遠へと向かうつもりである。



# とりあえずFDCについて

FDC (Floppy Disk Controller) とは、そのものずばりに、フロッピーディスクをコントロールするための LSI である。FDC の種類は、「単密度専用」と「倍密度/単密度両用」の二つに分かれるが、「単密度専用」はすでに過去の遺物なので、「倍密度/単密度両用」についてのみ書くことにする。

倍密度/単密度両用 FDC は、

- 1) FD1791 ファミリ
- 2)  $\mu$ PD765 ファミリ
- 3) その他

に分かれる。

FD1791 はウェスタン・デジタル社が開発した FDC で、X1/X1 turbo に使われている MB8877 (富士通製) は FD1791 ファミリに属する。それに対して  $\mu$ PD765 は日本電気の開発した FDC である。当然のことながら、二つのファミリには互換性などというものは存在しない (それぞれが書き込んだディスクはもちろんやりとりはできるが)。

ちなみに、FM シリーズの FDC は X1 と同じ MB8877、MZ シリーズは MB8876 を使っている。8877 と 8876 の違いは、バスラインが正論理か、負論理かの違いだけである。具体的には、データの 1/0 が反転するのである。すなわち、X1 でディスクに  $\&HFF=\&B11111111$  というデータを書き込み、そのディスクを MZ のドライブに差し込むと  $\&H00=\&H00000000$  が読み出せるというわけである。またディスクの表裏 (SIDE) も逆である。試してみると分かるが、X1 の第 0 レコード (シリンダ 0, サイド 0, セクタ 1) は MZ にとっては第 16 レコード (シリンダ 0, サイド 1, セクタ 1) である。逆に MZ の第 0 レコードは X1 にとっては第 16 レコードである。

本題に入る。表 9-5 が MB8877/8876 のコマンド、表 9-6~9-8 が各フラグの意味、表 9

表 9-5 MB8876/8877 のコマンド

タイプ		コマンド名称	動作	7 6 5 4 3 2 1 0	X1 での標準的な値
I	0	リストア	トラック 0 へ、ヘッドを移動する	0 0 0 0 h V r <sub>1</sub> r <sub>0</sub>	02 <sub>H</sub> =0 0 0 0 0 0 1 0
	1	シーク	所定のトラックへ、ヘッドを移動する	0 0 0 1 h V r <sub>1</sub> r <sub>0</sub>	1E <sub>H</sub> =0 0 0 1 1 1 1 0
	2	ステップ	ヘッドを 1 トラック移動する	0 0 1 u h V r <sub>1</sub> r <sub>0</sub>	3A <sub>H</sub> =0 0 1 1 1 0 1 0
	3	ステップ・イン	ヘッドを 1 トラック内側へ移動する	0 1 0 u h V r <sub>1</sub> r <sub>0</sub>	5A <sub>H</sub> =0 1 0 1 1 0 1 0
	4	ステップ・アウト	ヘッドを 1 トラック外側へ移動する	0 1 1 u h V r <sub>1</sub> r <sub>0</sub>	7A <sub>H</sub> =0 1 1 1 1 0 1 0
II	5	リードデータ	ディスクのデータ(データフィールド)を読む	1 0 0 m S E C 0	80 <sub>H</sub> =1 0 0 0 0 0 0 0
	6	ライトデータ	ディスク(データフィールド)へデータを書き込む	1 0 1 m S E C a <sub>0</sub>	A0 <sub>H</sub> =1 0 1 0 0 0 0 0
III	7	リードアドレス	ディスクの ID フィールドを読む	1 1 0 0 0 E 0 0	C0 <sub>H</sub> =1 1 0 0 0 0 0 0
	8	リードトラック	ディスクの 1 トラック分の全データを読む	1 1 1 0 0 E 0 0	E0 <sub>H</sub> =1 1 1 0 0 0 0 0
	9	ライトトラック	ディスクへ 1 トラック分の全データを書き込む	1 1 1 1 0 E 0 0	F0 <sub>H</sub> =1 1 1 1 0 0 0 0
IV	10	フォースインタラプト	割り込みを発生させる	1 1 0 1 l <sub>3</sub> l <sub>2</sub> l <sub>1</sub> l <sub>0</sub>	D0 <sub>H</sub> =1 1 0 1 0 0 0 0



-9, 9-10 がステータスレジスタの意味である。どーだ、これだけではちんぷんかんぷんだろう。そこで順に解説してゆくわけであるが、その前に表 9-11 がある。これが X1/X1 turbo での FDC 関係の I/O アドレスである。アミかけ部分は X1 turbo 用である。これは後でまたねっとりとやる。

表 9-6 TYPE I コマンド フラグ機能表

フラグ	機 能																		
u	トラックレジスタ更新フラグ(u)は、ヘッド移動に際し、トラックレジスタを更新することを指示する。 u = 1: トラックレジスタを更新する。 u = 0: トラックレジスタを更新しない。																		
h	ヘッドロードフラグ(h)は、コマンド実行開始時にヘッドをロードするか、ヘッドをメディアから離すかを指示する(X1では無意味)。 h = 1: コマンド実行開始時にヘッドロードする。 h = 0: コマンド実行開始時にヘッドを離す。																		
v	トラック照合フラグ(V)は、ヘッド移動後、ディスクのトラック番号とトラックレジスタの照合を行なうかを指示する。 V = 1: トラックの照合を行なう。 V = 0: トラックの照合は行なわない。																		
r <sub>1</sub> , r <sub>0</sub>	ステップレートフラグ(r <sub>1</sub> , r <sub>0</sub> )は、ステップパルス出力の間隔を指定する。 <table><tr><td></td><td colspan="2">ク ロ ッ ク</td></tr><tr><td>r<sub>1</sub> r<sub>0</sub></td><td>1MHz (X1 ノーマル)</td><td>2MHz (turbo の 2HD)</td></tr><tr><td>0 0</td><td>6ms</td><td>3ms</td></tr><tr><td>0 1</td><td>12ms</td><td>6ms</td></tr><tr><td>1 0</td><td>20ms</td><td>10ms</td></tr><tr><td>1 1</td><td>30ms</td><td>15ms</td></tr></table>		ク ロ ッ ク		r <sub>1</sub> r <sub>0</sub>	1MHz (X1 ノーマル)	2MHz (turbo の 2HD)	0 0	6ms	3ms	0 1	12ms	6ms	1 0	20ms	10ms	1 1	30ms	15ms
	ク ロ ッ ク																		
r <sub>1</sub> r <sub>0</sub>	1MHz (X1 ノーマル)	2MHz (turbo の 2HD)																	
0 0	6ms	3ms																	
0 1	12ms	6ms																	
1 0	20ms	10ms																	
1 1	30ms	15ms																	

表 9-7 TYPE II, III コマンド フラグ機能表

フラグ	機 能
m	マルチレコードフラグ(m)は連続セクタでリード/ライトするかを決める。 m = 1: 連続セクタ(セクタ番号が増加する方向)でリード/ライトを行なう。 m = 0: 単一セクタでリード/ライトを行なう。
S	サイドフラグ(S)はサイド番号指定に使う。 S = 1: サイド番号の LSB が 1 のとき一致したものとみなす。 S = 0: サイド番号の LSB が 0 のとき一致したものとみなす。 (このフラグは、C フラグが 1 のときのみ有効)
E	ディレイフラグ(E)は HLT 信号のサンプリングのタイミングを指定する(X1では無意味)。 E = 1: HLD(ヘッドロード)信号を "H" とした後 15ms 待ち、HLT 信号をサンプリングする。 E = 0: HLD 信号を "H" とした後、直ちに HLT 信号をサンプリングする。
C	サイド番号比較フラグ(C)はサイド番号と比較(チェック)するかどうかを指示する。 C = 1: サイド番号の比較を行なう。 C = 0: サイド番号の比較を行なわない。
a <sub>0</sub>	アドレスマークフラグ(a <sub>0</sub> )はデータアドレスマークに何を書くか指定する。 a <sub>0</sub> = 0: データアドレスマークに(FB) <sub>H</sub> (Data Mark)を書く。 a <sub>0</sub> = 1: データアドレスマークに(F8) <sub>H</sub> (Deleted Data Mark)を書く。



表 9-8 TYPE IV コマンド フラグ機能表

フラグ	機 能
I <sub>0</sub>	I <sub>0</sub> = 1 で READY 入力の立ち上がりで IRQ 発生 (IRQ = "H")。
I <sub>1</sub>	I <sub>1</sub> = 1 で READY 入力の立ち下がり IRQ 発生。
I <sub>2</sub>	I <sub>2</sub> = 1 でインデックスパルス検出時に IRQ 発生。
I <sub>3</sub>	I <sub>3</sub> = 1 で無条件で直ちに IRQ 発生

注意：X1/X1 turbo に使われている MB8877 の IRQ 信号は OPEN(どこにもつながっていない)だから、IRQ は何の機能も果たさない。

表 9-9 ステータスレジスタの各ビットの意味

コマンド \ ステータスレジスタ		bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
TYPE I	すべてのコマンド	NOT READY	WRITE PROTECT	HEAD ENGAGED	SEEK ERROR	CRC ERROR	TRACK00	INDEX	BUSY
TYPE II	リードデータ	NOT READY	0	RECODE TYPE	RECORD NOT FOUND	CRC ERROR	LOST DATA	DATA REQUEST	BUSY
	ライトデータ	NOT READY	WRITE PROTECT	WRITE FAULT	RECORD NOT FOUND	CRC ERROR	LOST DATA	DATA REQUEST	BUSY
TYPE III	リードアドレス	NOT READY	0	0	RECORD NOT FOUND	CRC ERROR	LOST DATA	DATA REQUEST	BUSY
	リードトラック	NOT READY	0	0	0	0	LOST DATA	DATA REQUEST	BUSY
	ライトトラック	NOT READY	WRITE PROTECT	WRITE FAULT	0	0	LOST DATA	DATA REQUEST	BUSY
TYPE IV	(他のコマンド実行中の場合)	(今まで実行していたコマンドのステータスビットと同様の意味)							0
	(実行中のコマンドのない場合)	NOT READY	WRITE PROTECT	HEAD ENGAGED	0	0	TRACK00	INDEX	0

表 9-10 ステータスの意味

コマンド	ステータス	意 味
TYPE I	NOT READY (bit 7)	NOT READY = 1 でディスクドライブが動作可能状態でないことを示す。
	WRITE PROTECT (bit 6)	WRITE PROTECT = 1 でディスクへの書き込みが禁止されていることを示す。
	HEAD ENGAGED (bit 5)	HEAD ENGAGED = 1 でヘッドがメディアに押しつけられていることを示す。
	SEEK ERROR (bit 4)	SEEK ERROR = 1 でベリファイ動作が成功しなかったことを示す。これは、ID フィールドが検出されなかったか、ID フィールドのトラック番号とトラックレジスタの内容が一致しないことによる。
	CRC-ERROR (bit 3)	CRC-ERROR = 1 で、ID フィールド読み出し時に読み出しエラーがあったことを示す。
	TRACK00 (bit 2)	TRACK00 = 1 でディスクのヘッドがトラック 0 の上にあることを示す。
	INDEX (bit 1)	INDEX = 1 でインデックスホールを検出したことを示す。



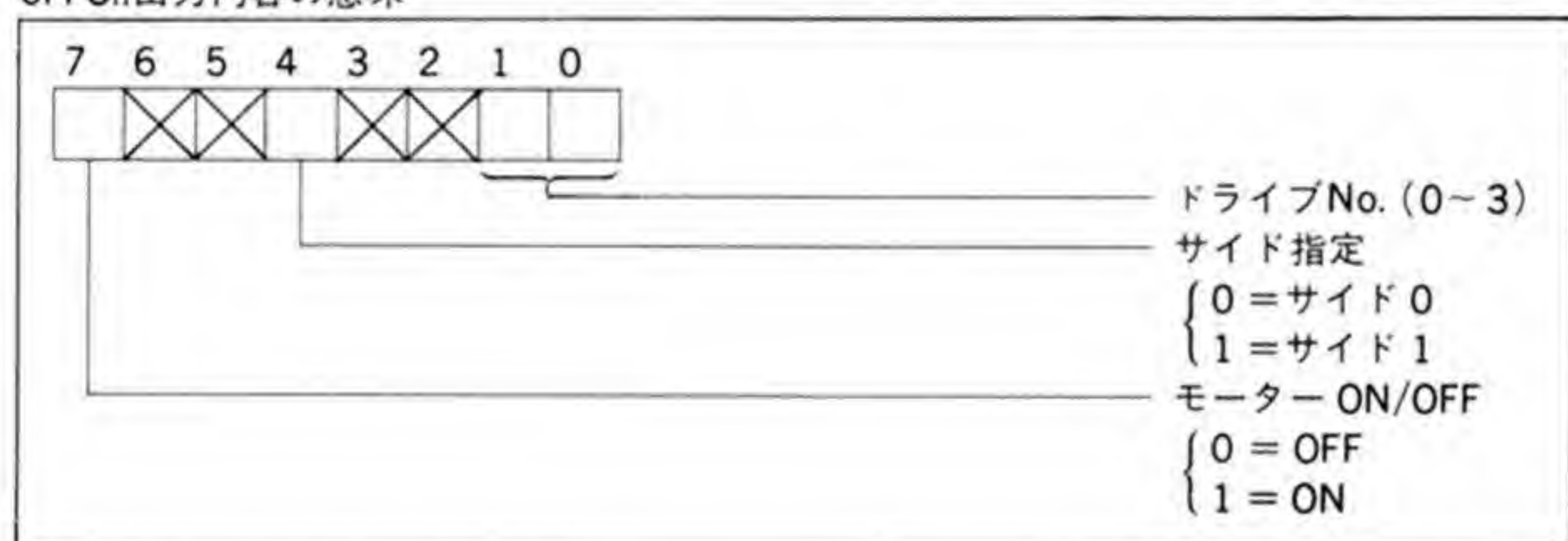
	BUSY (bit 0)	BUSY = 1 で FDC がコマンド動作中であることを示す。
TYPE II/III	NOT READY (bit 7)	NOT READY = 1 でディスクドライブが動作可能状態でないことを示す。
	WRITE PROTECT (bit 6)	WRITE PROTECT = 1 でディスクへの書き込みが禁止されていることを示す。
	RECORD TYPE /WRITE FAULT (bit 5)	リード動作のとき、RECORD TYPE の表示として、デリートッドデータマークのときセットされ、データマーク時リセットされる。ライト動作のとき、WRITE FAULT の表示として、書き込み動作が打ち切られたことを示す。
	RECORD NOT FOUND (bit 4)	RECORD NOT FOUND = 1 で指定されたトラック番号、サイド番号、セクタ番号を持ち、正しく読み出された ID フィールドがなかったことを示す。
	CRC ERROR (bit 3)	CRC ERROR = 1 でディスクから ID フィールドもしくはデータフィールドの読み出しにおいて、読み出しエラーがあったことを示す。
	LOST DATA (bit 2)	LOST DATA = 1 で所要時間内に DR の読み出し、もしくは書き込みが行なわれなかったデータがあったことを示す。
	DATA REQUEST (bit 1)	DATA REQUEST = 1 で、FDC がプロセッサに対して DR の読み出しもしくは書き込みを要求していることを示す。
	BUSY (bit 0)	BUSY = 1 で FDC がコマンド実行中であることを示す。

表 9-11 FDC 関係の I/O アドレス

0FF8H	ステータスレジスタ (STR)/コマンドレジスタ (CR)	IN/OUT
0FF9H	トラックレジスタ (TR)	IN/OUT
0FFAH	セクタレジスタ (SCR)	IN/OUT
0FFBH	データレジスタ (DR)	IN/OUT
0FFCH	FM 方式指定/ドライブ No., サイド, モーター ON	IN/OUT
0FFDH	MFM方式指定	IN
0FFEH	1.6M タイプ(2HD)指定	IN
0FFFH	500K(2D)・1M(2DD)切り換え	IN

の部分は turbo 用

0FFCH 出力内容の意味



## TYPE I の実習

FDC, MB8877 を実際に使ってみるのである。まず、TYPE I のコマンドは機械語でなく、BASIC からでも使えるのでちょうどよい肩慣らしである。ただし、FDC を誤って動かすとディスクの内容がパーになったりすることがしばしばあるので、書き換わってもよいディスク以外はセットしておかないように。

FDC を使う前にしなければならないのが、ドライブのセレクトとモーター ON である。これには 0FFCH 番地を使うのである。まずドライブ 1 ("1:") に壊れてもよいフォ



ーマット済みのディスクを入れて、BASIC から直接、

OUT &HFFC, &H81

と打ち込んでいただきたい。ドライブ1のLEDがつき、ディスクが回り出したはずである。不幸にしてドライブ0しか持っていない人は、&H81の代わりに&H80である。ドライブ0が回り出すであろう。冷たいようだが以下同様。

いつまで回していても仕方がないので、止めることにする。

OUT &HFFC, &H01 (もしくは&H00)

と打ち込むと、約1秒後にディスクが止まる。なぜ1秒後かという、これはディスクが回り出してから、読み書きできるようになるまでに時間がかかるからなのだ(わかるかな)。つまり、普通のディスクアクセス法として、一番自然な方法は、

- ① ディスクを回し始める
- ② ディスクが規定の速度(毎分300回転)になるまで待つ
- ③ 読んだり、書いたりする
- ④ ディスクを止める

となっている。

問題は、①～④をやった後で、すぐにまた①～④に飛んでくるケースが多いことにある。④で一旦ディスクが止まってしまっているから、(すぐさま)次にやってきたときに、またディスクがアクセス可能な状態になるまで待たなければならない。これでは時間の無駄が大きいので、④で「モーター OFF」の命令を受けても、すぐにはディスクを止めず、1秒間の遅延時間を置くのである。これによって、続けざまのディスクアクセスに時間の無駄が出ず、なおかつソフトウェアで面倒なことをしないで済むのである。めでたし、めでたし。

では、もう少し進んでみる。リスト9-1である。

リスト9-1 TYPE I 実習プログラム

```
100 OUT &HFFC,&H81      : 'MOTOR ON "1:"
110   GOSUB "WNBSY"
120 OUT &HFF8,&H2        : 'RESTORE (第0トラックヘシーク)
130   GOSUB "WNBSY"
140 OUT &HFFB,20         : 'データレジスタ←目的トラック番号
150 OUT &HFF9,0          : 'トラックレジスタ←現在のトラック番号
160 OUT &HFF8,&H1E       : 'SEEK
170   GOSUB "WNBSY"
180 OUT &HFF8,&H5A        : 'STEP IN
190   GOSUB "WNBSY"
200 OUT &HFFC,&H1
210 END
220 LABEL "WNBSY"
230 PRINT "]" ; SPACE$(3);
240 STAT=INP(&HFF8)      : 'GET STATUS BYTE
250 PRINT RIGHT$("0"+HEX$(STAT),2);SPACE$(2);
260 IF STAT AND &H81 THEN 240
270 PRINT ":" ; RETURN
```

まずはFFC<sub>H</sub>←81<sub>H</sub>で、ドライブ1のモーターをONする。GOSUB "WNBSY"で、ディスクが正しく定常状態になるまで待つ。次にFF8<sub>H</sub>番地(コマンドレジスタ)へ02<sub>H</sub>(表9-5のリストAコマンド)をOUTする。これで、ドライブ1のヘッドは第0トラック(一



番外側)へ移動してくれる。もちろん、ヘッドという重さのある「ブツ」が動くわけで多少の時間がかかるから、130行のGOSUB“WNBSY”で動作の完了を待ってやる。

140～170行は「シーク」である。このコマンドの使い方は、まず下準備として、FFB<sub>H</sub>番地(データレジスタ)に目的トラック番号をOUTし、FF9<sub>H</sub>番地(トラックレジスタ)へ現在のトラック位置をOUTする。次にコマンドレジスタへシークコマンド(1E<sub>H</sub>)を送るのである。以上のことをすると、ドライブは「クククッ」という音を立てて、目指すトラックへ動いてくれる。

その次はステップインで、これは要するに1トラックだけ内側にシークする。

さて、220行からのサブルーチンであるが、これはFF8<sub>H</sub>(ステータスレジスタ)を見張りつつ、MSBとLSB(第7ビットと第0ビット)がともに0になるのを待つものである。第7ビットは「NOT READY」、すなわち「まだ準備できてないよ」という意味。第0ビットは「BUSY」、すなわち「いま忙しいのっ!」もしくは「まだほかのことしてる最中でんがな。そないにせかさんといてや」という意味である。注意しておくが、I/OのFF8<sub>H</sub>番地は、

OUT したときはコマンドレジスタ

IN したときはステータスレジスタ

となっている。1番地2役なのだ。

このプログラムでは、ステータスの内容をチェックする意味で、いちいちプリントさせている。実行結果は、図9-4のようになるはずである。ちょっと見てみる。

図9-4 「リスト9-1」の実行例

J	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0
A0	A0	A0	A0	A2	A0	A0	A0	A0	A0	A0	A0	A0	A0	A0
A0	A0	20	:											
J	01	01	01	01	01	01	01	01	01	01	01	01	01	01
01	01	01	01	01	01	01	01	01	01	05	04	:		
J	21	21	21	21	21	21	21	21	21	21	21	21	21	21
21	21	21	21	21	21	21	21	21	21	21	21	20	:	
J	21	20	:											

まず“J”は、「WNBSYが始まったよ」という意味である。最初のやつは、モーターONから定常状態になるまで待っているわけだから、案の定A0<sub>H</sub>=10100000<sub>B</sub>で第7ビットが1である。いっしょに第5ビットも立っているが、これは「ヘッドがディスクに押し付けられている」ということを示している——はずだが、どっこい、実はX1ではこの信号は何の意味も持たないのであった。

ちょっと深入りするが、FDC⇄FDDの間にはHLDとHLTという信号がある。HLDはFDCからFDDへの「ヘッドをディスクに押し付けなさい」という信号である。HLTはFDDからFDCへの信号で「私ことFDDはディスクにヘッドを押し付けました。謹んでご報告いたします」というものである。しかしX1では、HLDはオープン、HLTは5Vレベルアップされているのだ。すなわち、HLD信号は何にも使われておらず、HLTは常に



High のままである。これはどういうことかということ、5.25 インチ以下のサイズのフロッピーディスクドライブでは一般的に「ヘッドは常にディスクに押し付けられているから」なのだ。X1 のディスクでもそうになっている。よって第5ビットは無視すべきである。

さて、図 9-4 の 2 行目の中程に「A2」というのが 1 か所出てくる。これは第1ビットが立っているのである。表 9-9 から「INDEX」と分かる。すなわち、FDC はこの瞬間にインデックスホールを検出したのだ。後はずーっと A0<sub>H</sub> で、最後に第7ビットが 0 になり「NOT READY ではない」すなわち「用意できたよーん」となって「WNBSY」からリターンするのである。言い遅れたが、このようなステータスになるのは、直前(最後)に FDC に送られたコマンドが TYPE I のコマンドだったからである。しかし、TYPE が何であっても「NOT READY」と「BUSY」は同じことだから、悩む必要はない。

次に「WNBSY」に来るのは、RESTORE (リストア：トラック 0 へのシーク) のときである。ステータスは 01<sub>H</sub> = 00000001<sub>B</sub> と、BUSY が並んでいる。最後の方は、05<sub>H</sub>、04<sub>H</sub> となっている。05<sub>H</sub> は表 9-9 から分かるように「TRACK 00 です & BUSY」である。ちょっと待て！ てめー、すでに TRACK 0 に来てるんなら、RESTORE 動作は終わってんじゃないか。それなのに BUSY とは何事だ、と一瞬逆上するが、次の 04<sub>H</sub> で FDC は BUSY を取り下げている。むーん。それにしても第5ビットが 0 だ——一体どーなっているのだ。ま、無視しておこう。

次がシークコマンド後の「WNBSY」である。おっと、21<sub>H</sub> = 00100001<sub>B</sub> だ。いやいや無視無視。えーと、こっちは平凡に BUSY が 0 になって終わりである。

最後がステップ・インである。これはシークと同じであるが、移動距離が短いので、あっという間に終わっている。

以上で図 9-4 の実感放送を終わる。

ではまず、シークコマンドについて少々補足しておく。シークコマンドでは、

データレジスタ ← 目的トラック番号

トラックレジスタ ← 現在のトラック番号

とセットするわけだが、どーしてこんな面倒なことをするのかということ、FDC が何台もの FDD を管理しているからなのだ。つまり、何台もの FDD を操作しているのだから、「現在のトラック番号」は X1 の場合最大で四つある。本来なら、トラックレジスタは移動後の「トラック番号」を保持しているので、1 台のドライブだけを動かしている場合は、目的トラック番号をデータレジスタにセットするだけで OK なのである。しかし現実ほとんどのシステムで FDD は 2 台以上あるから、DOS (もしくはディスク BASIC) は 4 バイトのワークエリアを持っていて、いちいちトラック番号を保持、書き込みしてやらなければならない。

トラックレジスタがトラック番号を持っていることを確認するためには、

```
175 PRINT INP(&HFF9)
```

```
195 PRINT INP(&HFF9)
```

を入れてやればよい。「20」、「21」と表示されるはずである。

次に大事な「フラグ」について説明する。リストアには、h, V, r<sub>1</sub>, r<sub>0</sub> の四つのフラグ



があり、各動作を指定している。まずは表 9-6 を見ると、

**h=ヘッドロードフラグ**

とある。これはさっき言ったように無意味である。次に、

**V=トラック照合フラグ**

である。これは実際にディスクを読んで、チェックするかどうかのフラグである。ディスクには「私は第△トラックの第×サイドの第○セクタです」という「ID」が書かれているのだ。これを読んで、ヘッドが本当に目的のトラックに達したかどうかのチェックを指定するフラグである。リスト 9-1 では 02<sub>H</sub>だから V=0 で、チェックは行なっていない。これについては、後で実験する。

次に r<sub>1</sub>, r<sub>0</sub> であるが、これは二つで一組で、00<sub>B</sub>~11<sub>B</sub> までの四つの場合があり、これはステップレートを指定するものである。ステップレートは表 9-6 に示してある「時間」である。なぜこんなものがあるかというと、これは「FDD のヘッドを動かす速さ」に機種ごとに差があるからなのだ（一般に高価な FDD ほど速い）。FDC は FDD に「ヘッドを動かさない」という信号を送るのだが、その信号は「方向（内側へか外側へか）」と「動け」だけなのだ。FDD は一般に **タコだから**、FDC の方でタイミングを取ってやらないといけない。そのタイミングを指定するのが r<sub>1</sub>, r<sub>0</sub> のステップレートなのである。たとえばリスト 9-1 のように r<sub>1</sub>=1, r<sub>0</sub>=0 を指定すると、FDC は FDD に対して 20 ms (0.02 秒) ごとに「動け」と指令を出すことになる。さて、実は X1 ではステップレートは別に 20 ms でなくてもよいのである。

3 インチ=3ms

turbo の内蔵=6ms

CZ-502F, 503F=6ms

古いタイプ（レバーのないやつ）=20 ms

などとなっている。

そこでリスト 9-2 が「ガリガリ」プログラムである。これは指定したステップレートで、ひたすらトラック 0 → 39 間を動くというものである。しつこくやったり、性能以上の速度で動かそうとすると故障の原因になるから、ほどほどにやること。性能を超えた場合は「32」とか「30」（第 4 ビット=シークエラーが 1）などが表示されるはずである。

後に残っているのは、STEP などの、

**u=トラックレジスタ更新フラグ**

である。これは「ヘッドの移動にともなって、トラックレジスタの値を増減させるか、させないか」のフラグである。リスト 9-1 で 180 行の &H5A を &H4A にして、

195 PRINT INP(&HFF9)

を追加すると「20」と表示されるはずである。実際には、ヘッドは第 21 トラック上にあるのだが、トラックレジスタはそうならないのである。

なぜこんなフラグがあるかというと、フロッピーディスクの始祖、IBM フォーマットには「欠陥トラック」というものがあるためなのだ。たとえば、ディスクをフォーマットしているときにどうしても正常に読み書きできないトラックを発見したとする。現在のパソ



リスト 9-2 ガリガリプログラム

```

100 INPUT "STEP RATE";S:IF (S<0) OR (S>3) THEN 100
110 OUT &HFFC,&H81      :'MOTOR ON "1:"
120     GOSUB"WNBSY"
130 OUT &HFF8,&H0+S      :'RESTORE (第0トラックヘシーク)
140     GOSUB"WNBSY"
150 OUT &HFFB,39         :'データレジスタ←目的トラック番号
160 OUT &HFF9,0          :'トラックレジスタ←現在のトラック番号
170 OUT &HFF8,&H1C+S     :'SEEK
180     GOSUB"WNBSY"
190 IF INKEY$(0)=" " THEN 130
200 OUT &HFFC,&H1
210 END
220 LABEL"WNBSY"
230 PRINT"]";SPACE$(3)
240 STAT=INP(&HFF8)      :'GET STATUS BYTE
250 PRINT RIGHT$("0"+HEX$(STAT),2);SPACE$(2);
260 IF STAT AND &H81 THEN 240
270 PRINT":":RETURN

```

コンではそんな場合には「エラーだよ、このディスクは使えません。ぴびっ」とメッセージを出して、別のディスクを使うように要求するのだが、IBM フォーマットでは、「交替用トラック」が2トラック分（本当はシリンダと表現すべきだが）用意されていて、多少の欠陥のあるディスクセットでも使えてしまうのである。ご丁寧にも、その欠陥のあるトラックを指定する方法も決まっている。この場合に、欠陥のあるトラックを飛ばしてフォーマットすると、実際のヘッドの位置と、ディスクに書かれている「ID」に違いが出てしまうことになる。そうすると、Vフラグでチェックしたときにシークエラーが出てしまうなどして都合が悪いので、uフラグを使って「つじつまを合わせる」のである。大昔は磁性体の品質が低かったり、ディスク（メディア）も高かったりしたから、そんなことをしたのだろうが、今はほとんど意味がないと言えるフラグである。しかし、IBM フォーマットがいかに「金科玉条」であるかのよい証拠でもある。

というところで、少々遊んでみることにする。まずはフォーマットされていない、まっさらのディスクを用意していただきたい。それをドライブ1に入れてもらう。その状態で、リスト 9-1 を実行すると、3 番目のシークコマンドのステータスで、最後に 30<sub>H</sub>が出るはずである。すなわちシークエラーである。これは別にフォーマットされているディスクでも、ステップ・インで、u=0, V=1（つまりコマンドは&H4E）とすると同じことが起こる。トラックレジスタの値は「20」なのに、Vフラグに従ってチェックしてみると、第「21」トラックだった、というわけである。

TYPE I コマンドで言い残したのは、STEP コマンドだけである。このコマンドを実行すると、ヘッドは内/外どちらに動くのかということであるが、「直前に動いた方」に動くということである。すなわち、FDC は「最後に動いた方向」を保持しているというわけである。



## TYPE II の実習

TYPE Iでは機械語を使う必要がなかったが、TYPE IIではさすがにそうはいかない。そこで、リスト 9-3、9-4、9-5である。例によってリスト 9-3は機械語プログラムのアセンブルリスト、リスト 9-4はダンプリスト、リスト 9-5はそれを BASIC から使うためのプログラムである。

BASIC は CZ-8FB01 の Ver 1.0

でなければならない。なぜかという、場合によっては E000<sub>H</sub> ~ F800<sub>H</sub> ぐらいまでをワークエリアに使ってしまうので、turbo BASIC では動かない(暴走するかもしれない)からである。

リスト 9-3 ディスクアクセスプログラム

```

                                .Z80
                                .PHASE   0DF00H
;
0FF8      CR      EQU      0FF8H    ;COMMAND REG.
0FF8      STR      EQU      0FF8H    ;STAT. REG.
0FF9      TR       EQU      0FF9H    ;TRACK REG.
0FFA      SCR      EQU      0FFAH    ;SECTOR REG.
0FFB      DR       EQU      0FFBH    ;DATA REG.
0FFC      MSDR     EQU      0FFCH    ;MOTOR,SIDE,DRIVE# SELECT
;
DF00      D5
DF01      1A
DF02      13
DF03      D5
DF04      21 DF1A
DF07      16 00
DF09      87
DF0A      5F
DF0B      19
DF0C      5E
DF0D      23
DF0E      56
DF0F      EB
DF10      D1
DF11      F3
DF12      CD DF19
;
DF15      FB
DF16      D1
DF17      12
DF18      C9
;
DF19      E9
;
PATCH:   JP        (HL)    ;CALL (HL)
;
;          Jump Table Area
;
;TYPE I
JTA:      DW        RSTR     ;RESTORE
           DW        SEEK     ;SEEK
           DW        STEP     ;STEP
           DW        STPIN    ;STEP IN
           DW        STPOT    ;STEP OUT
;
;TYPE II
           DW        READD     ;READ DATA
           DW        WRITD     ;WRITE DATA
;
;TYPE III
           DW        READI     ;READ ID
           DW        REDTR     ;READ TRACK
           DW        WRTTR     ;WRITE TRACK
;
;TYPE IV
DF2E      DFBC
           DW        FORI      ;FORCE INTERRUPT
;
;MOTOR,SIDE,DRIVE#

```



DF30	DFC6		DW	MSD	;MOTER, SIDE, DRIVE#
DF32	1A		RSTR:	LD	A, (DE) ;COMMAND
DF33	01 0FF8			LD	BC, CR ;BC=COMMAND REG.
DF36	ED 79			OUT	(C), A
DF38	CD DFD2			CALL	WNBSY ;WAIT READY
DF3B	C9			RET	
DF3C	EB		SEEK:	EX	DE, HL
DF3D	7E			LD	A, (HL) ;A=COMMAND
DF3E	23			INC	HL
DF3F	56			LD	D, (HL) ;D=TRACK# TO SEEK
DF40	23			INC	HL
DF41	5E			LD	E, (HL) ;E=CURRENT TRACK#
DF42	01 0FF9			LD	BC, TR
DF45	ED 59			OUT	(C), E ;SET CURRENT TRACK#
DF47	01 0FFB			LD	BC, DR
DF4A	ED 51			OUT	(C), D ;SET TRACK TO SEEK
DF4C	01 0FF8			LD	BC, CR
DF4F	ED 79			OUT	(C), A ;SEND SEEK COMMAND
DF51	CD DFD2			CALL	WNBSY ;WAIT 'NOT BUSY'
DF54	C9			RET	
DF55			STEP:		
DF55			STPIN:		
DF55	1A		STPOT:	LD	A, (DE) ;GET COMMAND
DF56	01 0FF8			LD	BC, CR
DF59	ED 79			OUT	(C), A
DF5B	CD DFD2			CALL	WNBSY
DF5E	C9			RET	
DF5F			READD:		
DF5F			READI:		
DF5F	CD DFEA		REDTR:	CALL	SETSCT ;A=COMMAND, SET SECTOR REG.
DF62	01 0FF8			LD	BC, CR ;CR=STR
DF65	D9			EXX	
DF66	01 0FFB			LD	BC, DR ;BC'=DATA REG.
DF69	2A DFFE			LD	HL, (BUFAD) ;HL'=BUFF ADDR.
DF6C	D9			EXX	
DF6D	ED 79			OUT	(C), A ;SET COMMAND
DF6F	CD DFE4			CALL	WAIT1
DF72	ED 78		RED1:	IN	A, (C) ;GET STATUS
DF74	0F			RRCA	;CHECK BUSY
DF75	30 0B			JR	NC, RED2 ;END READ
DF77	0F			RRCA	;CHECK DATA REQUEST
DF78	30 F8			JR	NC, RED1 ;NO REQUEST
DF7A	D9			EXX	
DF7B	ED 78			IN	A, (C) ;GET DATA
DF7D	77			LD	(HL), A ;STORE DATA
DF7E	23			INC	HL ;INC BUFF. ADDR.
DF7F	D9			EXX	
DF80	18 F0			JR	RED1 ;AGAIN
DF82	07		RED2:	RLCA	;BACK STAT (RRCA <-> RLCA)
DF83	C9			RET	
DF84			WRITD:		
DF84	CD DFEA		WRITR:	CALL	SETSCT ;A=COMMAND, SET SECTOR REG.
DF87	01 0FF8			LD	BC, CR ;CR=STR
DF8A	D9			EXX	
DF8B	01 0FFB			LD	BC, DR ;BC'=DATA REG.
DF8E	2A DFFE			LD	HL, (BUFAD) ;HL'=BUFF ADDR.
DF91	D9			EXX	
DF92	1E 00			LD	E, 00H ;COUNTER
DF94	16 02			LD	D, 02H ;MASK
DF96	ED 79			OUT	(C), A ;SET COMMAND
DF98	ED 78		FLOOP:	IN	A, (C) ;FIRST LOOP
DF9A	A2			AND	D
DF9B	C2 DFA4			JP	NZ, FIRST ;FIRST DATA REQ.
DF9E	1D			DEC	E ;DEC COUNTER
DF9F	C2 DF98			JP	NZ, FLOOP ;AGAIN
DFA2	18 06			JR	WRT1 ;QUIT FIRST LOOP
DFA4	D9		FIRST:	EXX	
DFA5	7E			LD	A, (HL) ;GET FIRST DATA
DFA6	ED 79			OUT	(C), A ;WRITE 1 BYTE
DFA8	23			INC	HL ;INC BUFF. ADDR.
DFA9	D9			EXX	
DFAA	ED 78		WRT1:	IN	A, (C) ;GET STATUS
DFAC	0F			RRCA	;CHECK BUSY
DFAD	30 0B			JR	NC, WRT2 ;END WRITE



DFAF	0F		RRCA		;CHECK DATA REQUEST
DFB0	30 F8		JR	NC,WRT1	;NO REQUEST
DFB2	D9		EXX		
DFB3	7E		LD	A,(HL)	;GET DATA
DFB4	ED 79		OUT	(C),A	;WRITE DATA
DFB6	23		INC	HL	;INC BUFF. ADDR.
DFB7	D9		EXX		
DFB8	18 F0		JR	WRT1	;AGAIN
DFBA	07	WRT2:	RLCA		;BACK STAT (RRCA <-> RLCA)
DFBB	C9		RET		
DFBC	1A	FOR1:	LD	A,(DE)	;GET COMMAND
DFBD	01 0FF8		LD	BC,CR	
DFC0	ED 79		OUT	(C),A	
DFC2	CD DFD2		CALL	WNBSY	
DFC5	C9		RET		
DFC6	1A	MSD:	LD	A,(DE)	;GET DATA
DFC7	01 0FFC		LD	BC,MSDR	
DFCA	ED 79		OUT	(C),A	;SET IT
DFCC	B7		OR	A	;MOTOR ON?
DFCD	F0		RET	P	;MOTOR OFF THEN RET
DFCE	CD DFD2		CALL	WNBSY	;WAIT READY
DFD1	C9		RET		
DFD2	C5	WNBSY:	PUSH	BC	;SAVE BC
DFD3	06 20		LD	B,20H	
DFD5	10 FE	WNBSY0:	DJNZ	WNBSY0	;WAIT
DFD7	01 0FF8	WNBSY1:	LD	BC,STR	;DASAI
DFDA	ED 78		IN	A,(C)	
DFDC	4F		LD	C,A	;SAVE STATUS
DFDD	E6 81		AND	81H	;CHECK
DFDF	20 F6		JR	NZ,WNBSY1	
DFE1	79		LD	A,C	;GET STATUS
DFE2	C1		POP	BC	;GET BC
DFE3	C9		RET		
DFE4	3E 07	WAIT1:	LD	A,7	;WAIT ROUTINE
DFE6	3D	WAIT2:	DEC	A	
DFE7	20 FD		JR	NZ,WAIT2	
DFE9	C9		RET		
DFEA	EB	SETSCT:	EX	DE,HL	
DFEB	7E		LD	A,(HL)	;A=COMMAND
DFEC	23		INC	HL	
DFED	56		LD	D,(HL)	;D=SECTOR#
DFEE	01 0FFA		LD	BC,SCR	;BC=SECTOR REG.
DFE1	ED 51		OUT	(C),D	;SET SECTOR#
DFE3	F5		PUSH	AF	;SAVE COMMAND
DFE4	CD DFD2		CALL	WNBSY	
DFE7	F1		POP	AF	;GET COMMAND
DFE8	C9		RET		
DFE9	00 00 00 00		DB	0,0,0,0,0	;DUMMY
DFED	00				
DFEE	E000	BUFAD:	DW	0E000H	;BUFFER ADDRESS
			END		

まずリスト 9-5 の使い方からである。基本的なパターンは、

番号(0~11), コマンド, パラメータ...

を CHR\$ で文字列に変換した後に USR0(〜)である。詳しくは表 9-12 に示す。11 番だけは FDC へのコマンドではなく、FFC<sub>H</sub> 番地へのデータであることに注意。

なお、このプログラムは手抜きのお手本みたいなもので、ちゃんとしたエラー処理をしていない。たとえばディスクを抜いて、ドライブのモーターを ON にしようとする、帰ってこないのである。そのときは心静かにリセットスイッチを押していただきたい。

では、おもむろにリスト 9-3 の解説を始める。



リスト 9-4 「リスト 9-3」のダンプリスト(念のため: チェックには付録 A を参照のこと)

DF00	D5	1A	13	D5	21	1A	DF	16	:	07
DF08	00	87	5F	19	5E	23	56	EB	:	C1
DF10	D1	F3	CD	19	DF	FB	D1	12	:	67
DF18	C9	E9	32	DF	3C	DF	55	DF	:	12
DF20	55	DF	55	DF	5F	DF	84	DF	:	09
DF28	5F	DF	5F	DF	84	DF	BC	DF	:	7A
DF30	C6	DF	1A	01	F8	0F	ED	79	:	2D
DF38	CD	D2	DF	C9	EB	7E	23	56	:	29
DF40	23	5E	01	F9	0F	ED	59	01	:	D1
DF48	FB	0F	ED	51	01	F8	0F	ED	:	3D
DF50	79	CD	D2	DF	C9	1A	01	F8	:	D3
DF58	0F	ED	79	CD	D2	DF	C9	CD	:	89
DF60	EA	DF	01	F8	0F	D9	01	FB	:	A6
DF68	0F	2A	FE	DF	D9	ED	79	CD	:	22
DF70	E4	DF	ED	78	0F	30	0B	0F	:	81
DF78	30	F8	D9	ED	78	77	23	D9	:	D9
-----										
SUM:	69	F3	1C	A0	7A	AD	85	E2	43E7	

DF80	18	F0	07	C9	CD	EA	DF	01	:	6F
DF88	F8	0F	D9	01	FB	0F	2A	FE	:	13
DF90	DF	D9	1E	00	16	02	ED	79	:	54
DF98	ED	78	A2	C2	A4	DF	1D	C2	:	2B
DFA0	98	DF	18	06	D9	7E	ED	79	:	52
DFA8	23	D9	ED	78	0F	30	0B	0F	:	BA
DFB0	30	F8	D9	7E	ED	79	23	D9	:	E1
DFB8	18	F0	07	C9	1A	01	F8	0F	:	FA
DFC0	ED	79	CD	D2	DF	C9	1A	01	:	C8
DFC8	FC	0F	ED	79	B7	F0	CD	D2	:	B7
DFD0	DF	C9	C5	06	20	10	FE	01	:	A2
DFD8	F8	0F	ED	78	4F	E6	81	20	:	42
DFE0	F6	79	C1	C9	3E	07	3D	20	:	9B
DFE8	FD	C9	EB	7E	23	56	01	FA	:	A3
DFF0	0F	ED	51	F5	CD	D2	DF	F1	:	B1
DFF8	C9	00	00	00	00	00	00	E0	:	A9
-----										
SUM:	6A	7F	EE	56	A4	E0	A9	89	EC8E	

リスト 9-5 ディスクアクセスサンプル

```

100 CLEAR &HDF00
110 MEM$(&HDF00,16)=HEXCHR$("D5 1A 13 D5 21 1A DF 16 00 87 5F 19 5E 23 56 EB")
120 MEM$(&HDF10,16)=HEXCHR$("D1 F3 CD 19 DF FB D1 12 C9 E9 32 DF 3C DF 55 DF")
130 MEM$(&HDF20,16)=HEXCHR$("55 DF 55 DF 5F DF 84 DF 5F DF 5F DF 84 DF BC DF")
140 MEM$(&HDF30,16)=HEXCHR$("C6 DF 1A 01 F8 0F ED 79 CD D2 DF C9 EB 7E 23 56")
150 MEM$(&HDF40,16)=HEXCHR$("23 5E 01 F9 0F ED 59 01 FB 0F ED 51 01 F8 0F ED")
160 MEM$(&HDF50,16)=HEXCHR$("79 CD D2 DF C9 1A 01 F8 0F ED 79 CD D2 DF C9 CD")
170 MEM$(&HDF60,16)=HEXCHR$("EA DF 01 F8 0F D9 01 FB 0F 2A FE DF D9 ED 79 CD")
180 MEM$(&HDF70,16)=HEXCHR$("E4 DF ED 78 0F 30 0B 0F 30 F8 D9 ED 78 77 23 D9")
190 MEM$(&HDF80,16)=HEXCHR$("18 F0 07 C9 CD EA DF 01 F8 0F D9 01 FB 0F 2A FE")
200 MEM$(&HDF90,16)=HEXCHR$("DF D9 1E 00 16 02 ED 79 ED 78 A2 C2 A4 DF 1D C2")
210 MEM$(&HDFA0,16)=HEXCHR$("98 DF 18 06 D9 7E ED 79 23 D9 ED 78 0F 30 0B 0F")
220 MEM$(&HDFB0,16)=HEXCHR$("30 F8 D9 7E ED 79 23 D9 18 F0 07 C9 1A 01 F8 0F")
230 MEM$(&HDFC0,16)=HEXCHR$("ED 79 CD D2 DF C9 1A 01 FC 0F ED 79 B7 F0 CD D2")
240 MEM$(&HDFD0,16)=HEXCHR$("DF C9 C5 06 20 10 FE 01 F8 0F ED 78 4F E6 81 20")
250 MEM$(&HDFE0,16)=HEXCHR$("F6 79 C1 C9 3E 07 3D 20 FD C9 EB 7E 23 56 01 FA")
260 MEM$(&HDFF0,16)=HEXCHR$("0F ED 51 F5 CD D2 DF F1 C9 00 00 00 00 00 00 E0")
270 DEFUSR0=&HDF00
280 D$=USR0(CHR$(11,&H81)) : 'MOTOR ON
290 D$=USR0(CHR$(0,&H2)) : 'RESTORE
300 D$=USR0(CHR$(1,&H1E,2,0)) : 'SEEK
310 'D$=USR0(CHR$(2,&H3A)) : 'STEP
320 'D$=USR0(CHR$(3,&H5A)) : 'STEP IN
330 'D$=USR0(CHR$(4,&H7A)) : 'STEP OUT
340 D$=USR0(CHR$(5,&H80,1)) : 'READ DATA
350 'D$=USR0(CHR$(6,&HA0,1)) : 'WRITE DATA
360 'D$=USR0(CHR$(7,&HC0,1)) : 'READ ADD.
370 'D$=USR0(CHR$(8,&HE0,1)) : 'READ TRACK
380 'D$=USR0(CHR$(9,&HF0,1)) : 'WRITE TRACK
390 'D$=USR0(CHR$(10,&HD0)) : 'FORCE INT.
400 D$=USR0(CHR$(11,&H1)) : 'MOTOR OFF
410 END

```

DF00<sub>H</sub>～DF19<sub>H</sub>は、0～11番のうちの、どこへ飛び込むかを計算している。最初に DE レジスタを PUSH しているのは、後々ステータスを返すためである。DF11<sub>H</sub>に「DI」が割込みを禁止しているが、これはディスクリード/ライトのようにタイミングが大事なプログラムでは必要なことである。DF12<sub>H</sub>では DF19<sub>H</sub>の「JP (HL)」とともに、「CALL (HL)」に相当することをやらせている。実際のルーチンは後回しにして、DF15<sub>H</sub>～DF18<sub>H</sub>を説明する。ディスクにアクセスして帰ってきたときに、A レジスタは最後に読み出したステータスレジスタの値を持っているようにしてある。その値を DE レジスタの指すアドレスにストアすると、このプログラムで呼び出した BASIC で、



表 9-12 「リスト 9-5」の使い方

コ マ ン ド	番号	第 2 パラメータ	第 3 パラメータ	第 4 パラメータ
リストア	0	コマンド	—	—
シーク	1	コマンド	目的シリンダ No.	現在のシリンダ No.
ステップ	2	コマンド	—	—
ステップ・イン	3	コマンド	—	—
ステップ・アウト	4	コマンド	—	—
リードデータ	5	コマンド	セクタ No.	—
ライトデータ	6	コマンド	セクタ No.	—
リードアドレス	7	コマンド	ダミー	—
リードトラック	8	コマンド	ダミー	—
ライトトラック	9	コマンド	ダミー	—
フォースインタラプト	10	コマンド	—	—
モーター, サイド, ドライブナンバー	11	データ	—	—

D\$=USR0(～)

となっていると、D\$の先頭の文字の ASCII コードとして受け取ることができる。便利、便利。

では、実行ルーチンの方であるが、たとえば DF32<sub>H</sub>の RESTORE の実行ルーチンでは、「LD A, (DE)」によって、FDC に送るコマンドを A レジスタに拾い上げている。それをコマンドレジスタ (CR) に OUT して、WNBSY をコールする。よーするにこれは前述の BASIC で書いたプログラムと同じである。SEEK, STEP などと同様。

次に DF5F<sub>H</sub>からの READD (リードデータ), READI (リードアドレス), REDTR (リードトラック) の 3 種混合ルーチンである。恐ろしいことにこれら三つのルーチンは完全に同じなのである。その結果、READI, REDTR では、必要がないのにセクタ番号を指定しているのである。実害はないから、大丈夫である。まず、「CALL SETSCT」で A レジスタにコマンドを拾い上げ、なおかつセクタレジスタ (SCR) にセクタ番号を OUT している。その先が少々複雑なのだが、BC レジスタにステータスレジスタ (SCR=CR) のアドレスをセットし、EXX で裏レジスタにして、BC' にデータレジスタ (DR), HL' にバッファアドレスをセットしている。ディスクのリード/ライトのためには、二つの違う I/O アドレスに高速にアクセスしなければならないのでこうしたのだが、STR と DR の上位アドレスはともに 0F<sub>H</sub>だから、どうしても裏レジスタを使わなければならないわけではない。結局は趣味の問題である。その後はコマンドを CR に OUT して、少々待つ。なぜかよく知らないが、とにかく待つ。それからレジスタを表にしたり、裏にしたりしつつ、データを受け取るのである。DF82<sub>H</sub>番地の RLCA は右にローテートしたステータスを左にローテートし直して、元に戻すものである。

次に DF84<sub>H</sub>からの書き込む方であるが、DF98<sub>H</sub>～DFA9<sub>H</sub>間は、1 バイトだけ書くルーチンである。なんでこんなことをしているかというと、ライトトラックのためなのだ。なぜかこうしないとライトトラックが成功しないのである。理由はよく分からないが動け



ばよいのだ。ほかは基本的にリードルーチンと同じである。

さて、リスト 9-5 を実行すると、ドライブ 1 に対してリストア動作した後 (290 行)、第 2 トラックヘシークし (300 行)、第 1 セクタをリードし (340 行)、モーターを OFF する。データを読み込むアドレスは、DFFE<sub>H</sub> 番地からの 2 バイトに格納されているアドレスで、リスト 9-5 では E000<sub>H</sub> になっている (260 行の右端)。D\$ の先頭バイトには、最後に読み込んだステータスレジスタの値が入っているから、

PRINT ASC (D\$)

で、エラーチェックもできるようになっている。そこで早速リードデータの代わりに、370 行のリードトラックを実行してみていただきたい。「D\$=～」の代わりに、「D1\$=～」とでもしておくべきだろう。実行後、PRINT ASC (D1\$) とすると、「4」が表示されるはずである。ステータス表 (表 9-9) を見ると、なんと「LOST DATA」である。これは結局どーゆーことかという、「さっさと読み出さなかったから、データの取りこぼしがあるかもしれないよ」ということである。つまり、早い話がエラーである。

話せばわかる！ X1 では、どうしてもこうなるのである。これは仕方がないのである。私のプログラムのせいではないのだ。しかしここにも一筋の光明があるのだ。X1 turbo ならば、「LOST DATA」にはなるが、充分にデータは読み出せているのである。turbo 以外の機種では (全部を確認したわけではないが)、データがビットずれを起こしきって、使いものにならない。

しかし、リードトラックよりも大事なことがあるのだ (と逃げる)。まずは TYPE II コマンドのフラグについて説明しておく。

**m=マルチレコードフラグ**

これは、複数のレコードを一度に読み書きするときに使うフラグである。後で実験する。

**S=サイドフラグ**

これは C フラグといっしょに使うもので、読み出したディスクのサイドが 0 であるべきか、それとも 1 であるべきかを指定する。

**E=ディレイフラグ**

これは、ヘッドがディスクに押し付けられたかどうかの信号のサンプリングタイミングを指定するもの。X1 では意味がない。

**C=サイド番号比較フラグ**

これは V フラグがトラック (シリンダ) 番号を比較するかどうかのフラグであるのに対して、サイド番号を比較するかどうかのフラグである。しつこく言うが、S フラグといっしょに使う。

**a<sub>0</sub>=アドレスマークフラグ**

これはライトデータのときだけ指定できるものである。こいつに関してはサンプルを示した方が早いだろう。

では、TYPE II の実習である。図 9-5、9-6、9-7、9-8 である。

まずは m フラグをいじってみる。リスト 9-5 の 280 行からを図 9-5 のように書き換えて



いただきたい。まずは 280 行でモーター ON, ドライブは "1:" でサイドは 0 番である。リストアして, m=1 で第 1 セクタから読み始めている。325 行で, それぞれのステータスレジスタを表示している。その値は,

リストア命令→4 (TRACK 00=1)

リードデータ命令→16 (RECORD NOT FOUND)

となっている。モニタに飛んでダンプしてみると, しっかりと 16 セクタ分が読み込まれているはずである。そこで, BASIC に帰り,

PRINT INP (&HFFA)

として, セクタレジスタの値を表示してみると「17」である。すなわち FDC は m フラグ=1 に従って, 第 1 セクタから第 16 セクタまでを読み, 挙げ句の果てに第 17 セクタまでを読もうとしたが, 「ない袖は振れない」の法則に従って, 当然のごとく失敗して, 「RECORD NOT FOUND」を起こしたというわけである。この m フラグを使うと, 連続セクタを読むという点ではなかなか高速なのだが, 惜しむらくは「いつ終わるのか指定できない」という欠点があるのだ。それで FDC は「第 17 セクタはどこだっ!？」と, 少しの間探し回るので, 結局は時間をくってしまう。よって, 正しく使うには, 必要なだけを読んだかどうかをチェックしてやって, 適当なところで, TYPE IV のフォースインタラプトを使ってコマンドを打ち切ってやらなければならない。なかなか世話のやける m フラグであった。

図 9-5 m フラグの例

m FLAG		
280	D\$=USR0(CHR\$(11,&H81))	: 'MOTOR ON
290	D0\$=USR0(CHR\$(0,&H2))	: 'RESTORE
300	D5\$=USR0(CHR\$(5,&H90,1))	: 'READ DATA
310	'D6\$=USR0(CHR\$(6,&HA0,1))	: 'WRITE DATA
320	D\$=USR0(CHR\$(11,&H1))	: 'MOTOR OFF
325	PRINT ASC(D0\$),ASC(D5\$),ASC(D6\$)	
330	END	
	4	16 0

次に S, C フラグである。図 9-6 は, 平和な例で, 280 行でサイド 0 を指定しているから, S=0 (サイドは 0 ですか?), C=1 (チェックしてください) によって, ステータスは 0 という値になる。この逆が図 9-7 で, 280 行でサイド 0 を指定しているのに, S=1 (サイドは 1 ですか?), C=1 (チェックしてください) としているので, しっかりと 16=RECORD NOT FOUND が出ている。この場合はデータの読み込みは行なわれない。

図 9-6 S, C フラグの例 その1

S, C FLAG...S=0, C=1		
280	D\$=USR0(CHR\$(11,&H81))	: 'MOTOR ON



```

290 D0$=USR0(CHR$(0,&H2))      : 'RESTORE
300 D5$=USR0(CHR$(5,&H82,1))    : 'READ DATA
310 'D6$=USR0(CHR$(6,&HA0,1))    : 'WRITE DATA
320 D$=USR0(CHR$(11,&H1))       : 'MOTOR OFF
325 PRINT ASC(D0$),ASC(D5$),ASC(D6$)
330 END

```

4            0            0

図 9-7 S, C フラグの例 その2

S, C FLAG...S=1, C=1

```

280 D$=USR0(CHR$(11,&H81))      : 'MOTOR ON
290 D0$=USR0(CHR$(0,&H2))      : 'RESTORE
300 D5$=USR0(CHR$(5,&H8A,1))    : 'READ DATA
310 'D6$=USR0(CHR$(6,&HA0,1))    : 'WRITE DATA
320 D$=USR0(CHR$(11,&H1))       : 'MOTOR OFF
325 PRINT ASC(D0$),ASC(D5$),ASC(D6$)
330 END

```

4            16            0

$a_0$  フラグの実習が図 9-8 である。 $a_0$  フラグ=1 で、ライトデータを実行すると、そのセクタにデリートアドレスマークが書かれてしまうのだ。具体的にどこかというのは後程やるのである。これを検出するには、そのセクタを読むだけでよい。図 9-8 のように、ステータスレジスタが 32 となり、第 5 ビット (RECORD TYPE) が 1 になる。この場合にデータはちゃんと読まれるから、特にチェックしない限りデリートアドレスマークが書かれていようがいまいが、同じことである。では、なぜこのようになっているのかというと、例によって「IBM に聞いてくれ」なのであった。

図 9-8  $a_0$  フラグの例

$a_0$  FLAG

```

280 D$=USR0(CHR$(11,&H81))      : 'MOTOR ON
290 D0$=USR0(CHR$(0,&H2))      : 'RESTORE
300 D6$=USR0(CHR$(6,&HA1,1))    : 'WRITE DATA
310 D5$=USR0(CHR$(5,&H80,1))    : 'READ DATA
320 D$=USR0(CHR$(11,&H1))       : 'MOTOR OFF
325 PRINT ASC(D0$),ASC(D5$),ASC(D6$)
330 END

```

4            32            0

## TYPE III

MB8877 の TYPE III コマンドにはリードアドレス、リードトラック、ライトトラックの



三つがあるわけである。そこでまずは、リードアドレスからである。

リードアドレスとは何かとつうと、ディスクに書かれている「ID フィールド」と呼ばれる部分を読み出すということである。正しくフォーマットされたディスクには、各セクタごとに ID フィールドとデータフィールドが1個ずつあるのだ。念のために言うと、「フィールド」とは「領域」とか「区画」という意味である。要するに平たく言えば、ID フィールドとはセクタの「ラベル」みたいなもので、データフィールドは「中身」である。

さて、ID フィールドの読み方であるが、リスト 9-6 である。縁起ものだから、リスト 9-5 と同じ形式で載せておくことにする。RUN した後にモニタに飛んで、E000<sub>H</sub>からをダンプすると最初の6バイトが大体図 9-9 のようになっているはずである。この内訳は、

- 0 バイト目=シリンダ番号
- 1 バイト目=サイド番号
- 2 バイト目=セクタ番号
- 3 バイト目=セクタ長 (0~3)
- 4 バイト目=CRC 上位
- 5 バイト目=CRC 下位

となっている。これらはすべて、フォーマット時に書き込まれたものである。セクタ番号は、最初に出会ったセクタのものである。よって、このままでは1~16番のうちどれが来るかは運まかせである。

セクタ長は、

- 0 → 1 セクタ当たり 128 バイト
- 1 → 1 セクタ当たり 256 バイト
- 2 → 1 セクタ当たり 512 バイト
- 3 → 1 セクタ当たり 1024 バイト

リスト 9-6 ID フィールドを読み出す場合の「リスト 9-5」の変更点100~270行は同じ

```
280 D$=USR0(CHR$(11,&H81))      : 'MOTOR ON
290 D$=USR0(CHR$(0,&H2))        : 'RESTORE
300 'D$=USR0(CHR$(1,&H1E,2,0))  : 'SEEK
310 'D$=USR0(CHR$(2,&H3A))      : 'STEP
320 'D$=USR0(CHR$(3,&H5A))      : 'STEP IN
330 'D$=USR0(CHR$(4,&H7A))      : 'STEP OUT
340 'D$=USR0(CHR$(5,&H80,1))    : 'READ DATA
350 'D$=USR0(CHR$(6,&HA0,1))    : 'WRITE DATA
360 D$=USR0(CHR$(7,&HC0,1))     : 'READ ADD.
370 'D$=USR0(CHR$(8,&HE0,1))    : 'READ TRACK
380 'D$=USR0(CHR$(9,&HF0,1))    : 'WRITE TRACK
390 'D$=USR0(CHR$(10,&HD0))     : 'FORCE INT.
400 D$=USR0(CHR$(11,&H1))       : 'MOTOR OFF
410 END
```

図 9-9 ID フィールドの内容

:E000=00 00 02 01 AF 5F 00 00 /.....7\_...



を指定している。普通は1セクタ256バイトだから「1」になっているはずである。最後の2バイトのCRCは、早い話がチェックサムみたいなものである。読み出すときのチェック用で、一定の方法で計算が行なわれる。FDCが勝手に計算してくれるので、別に心配する必要はない。もちろん、計算が合わなければ、「CRC ERROR」が発生するわけである。

というところで、もう一度図9-9を見ると、最初の2バイトは00<sub>H</sub>, 00<sub>H</sub>で、つまりは第0シリンダの第0サイドなのである。そして、第2セクタに最初に出会った。そのセクタのタイプは1だから、ありふれた容量256バイトのフォーマットである。最後の2バイトのCRCのAF<sub>H</sub>, 5F<sub>H</sub>は、CRCというわけである。

ここでわれわれその筋探検隊は、フロッピーディスクの深淵へと向かうのである。それはどーゆーことかという、「ディスクの中で各セクタはどのように並んでいるのだろう」という疑問である。普通に考えるなら、1, 2, 3, …… , 16であるが、別にそうでなくてもよいはずである。そこで考えてみると、たしかTYPE Iのコマンドを実行した直後ならINDEXホールを検出できるつつうことに思い当たるのであった。よって、インデックスホールを検出した後で、リードアドレスを立て続けに16回行なえば1周分の各セクタのIDフィールドが読めてしまうのである。

しかし、悲しいことにBASICでそれをやっても速度が追いつかないのである。結局はまたもや機械語を組まなければならない。うーむとつぶやきつつリスト9-7とリスト9-8である。

リスト9-7 IDフィールドの連続読み込み

		.Z80	
		.PHASE	0DE00H
0FF8		CR EQU	0FF8H ;COMMAND REG.
0FF8		STR EQU	0FF8H ;STAT. REG.
0FFB		DR EQU	0FFBH ;DATA REG.
DE00	F3	START: DI	
DE01	01 0FF8	LD	BC,CR ;CR=STR
DE04	ED 78	WINDEX: IN	A,(C)
DE06	E6 02	AND	02H ;CHECK INDEX
DE08	28 FA	JR	Z,WINDEX
DE0A	1A	LD	A,(DE)
DE0B	57	LD	D,A ;D=COUNTER
DE0C	D9	EXX	
DE0D	01 0FFB	LD	BC,DR ;DATA REG.
DE10	2A DE37	LD	HL,(BUFAD) ;BUFF ADD.
DE13	D9	EXX	
DE14	3E C0	AGAIN: LD	A,0C0H ;READ AD. COM.
DE16	ED 79	OUT	(C),A ;SEND COMMAND
DE18	3E 07	LD	A,7
DE1A	3D	WAIT: DEC	A
DE1B	20 FD	JR	NZ,WAIT
DE1D	ED 78	RED1: IN	A,(C)
DE1F	0F	RRCA	;CHECK BUSY
DE20	30 0B	JR	NC,RED2 ;END READ
DE22	0F	RRCA	;CHECK DATA REQ.
DE23	30 F8	JR	NC,RED1 ;NO REQ.
DE25	D9	EXX	
DE26	ED 78	IN	A,(C)
DE28	77	LD	(HL),A ;STORE DATA
DE29	23	INC	HL
DE2A	D9	EXX	



DE2B	18 F0		JR	RED1
		;		
DE2D	07	RED2:	RLCA	;BACK STAT
DE2E	D9		EXX	
DE2F	77		LD	(HL),A ;STORE STATUS
DE30	23		INC	HL
DE31	D9		EXX	
DE32	15		DEC	D ;DEC COUNTER
DE33	20 DF		JR	NZ,AGAIN
		;		
DE35	FB		EI	
DE36	C9		RET	
		;		
DE37	E000	BUFAD:	DW	0E000H
		;		
			END	

リスト 9-8 ID フィールドを連続読み込み表示する

270	'100 - 270 キウ ハ リスト9-5 ノモノヲ ツカウノデアル
280	CLEAR &HDE00
290	MEM\$(&HDE00,16)=HEXCHR\$("F3 01 F8 0F ED 78 E6 02 28 FA 1A 57 D9 01 FB 0F")
300	MEM\$(&HDE10,16)=HEXCHR\$("2A 37 DE D9 3E C0 ED 79 3E 07 3D 20 FD ED 78 0F")
310	MEM\$(&HDE20,16)=HEXCHR\$("30 0B 0F 30 F8 D9 ED 78 77 23 D9 18 F0 07 D9 77")
320	MEM\$(&HDE30,16)=HEXCHR\$("23 D9 15 20 DF FB C9 00 E0 00 00 00 00 00 00")
330	DEFUSR1=&HDE00
340	COUNT=20
350	TR=0
360	D\$=USR0(CHR\$(11,&H81)) : 'MOTOR ON
370	D\$=USR0(CHR\$(0,&H2)) : 'RESTORE
380	D\$=USR0(CHR\$(1,&H1E,TR,0)) : 'SEEK
390	D\$=USR1(CHR\$(COUNT)) : 'READ ADD.
400	FOR I=&HE000 TO &HE000+7*(COUNT-1) STEP 7
410	FOR J=I TO I+(7-1)
420	PRINT RIGHT\$("0"+HEX\$(PEEK(J)),2); " ";
430	NEXT:PRINT
440	NEXT
450	D\$=USR0(CHR\$(11,&H1)) : 'MOTOR OFF
460	END

リスト 9-7 は、まず FDC のステータスレジスタを IN して、第 1 ビット (=INDEX) が 1 になるまで待つ。その後、指定された回数だけバシバシとリードアドレスを実行して、最後の 1 バイトを読んだときのステータスといっしょにメモリに格納する。説明はこんなものでよいだろう。リスト 9-8 を見ていただきたい。これはリスト 9-7 の使い方である。100~270 行には、リスト 9-6 ( 9-5 ) の 100~270 行を必要とする。

やっていることは単純で、まず 360 行でモーターを ON して、370 行でとりあえずリストアを実行してヘッドを第 0 トラックへ持ってくる。380 行ではお望みのトラックまでシークしている。トラック番号は TR に入っている。そこですかさず、

D\$=USR1(CHR\$(読み出す回数))

である。最後のステータスもメモリに格納するから、1 回当たり 7 バイトのデータが取れるわけである。400 行からはそれを表示している。くれぐれも注意するが、USR1 の直前には TYPE I のコマンドを実行しておくこと。さもなくばリセットスイッチのお世話になってしまうのである。

ここで研究発表をするわけだが、それは図 9-10 である。HuBASIC の場合は平和に 1~16 が順に並んでいる。X1 CP/M は、第 0 と第 1 シリンダは HuBASIC と同じ、第 2 シリンダからは、図 9-10 に書いてあるようになっている。MZ-2000/2200 の CP/M では全シリンダにおいて図 9-10 のようになっている。



図 9-10 セクタの並び方

HuBASIC																
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
X1 CP/M (TRACK 2-39)																
1	14	11	8	5	2	15	12	9	6	3	16	13	10	7	4	
MZ CP/M																
1	3	5	7	9	11	13	15	2	4	6	8	10	12	14	16	

そこで、どーしてこうなっているかである。

一般には第1セクタを読み書きした後は第2セクタを読み書きするわけであるが、CP/Mのようにハードウェアと密着していないソフトウェアなどは、その2回の読み書きの間に別の処理が入り、その間にディスクが少々回ってしまうのである。つまり、1～16まで順に並んでいると、「第1セクタを読み終え、次に第2セクタを読もうとしても、すでに第2セクタはヘッドの位置を通り過ぎてしまっている」のだ。そうすると、ディスクがもう1回転するまで待たねばならない。前述のように、5インチや3インチの2D、2DDのディスクは1分間に300回転、すなわち1秒間に5回転しているから、1周分といえど0.2秒である。はっきり言って、これはとても長い時間なのだ。そこで、あらかじめ第1セクタと第2セクタの間を離しておくわけである。もちろんこれは第2セクタと第3セクタの間でも同じことである。第nセクタから何個目に第n+1セクタがあるかを数えてみると(もちろん16→0と巡回するんだよ)、

HuBASIC → 1個

X1 CP/M → 5個

MZ CP/M → 8個

となっている。この1, 5, 8, の数字はスキューファクタと呼ばれる。これと同じような意味を持つものにインターリーブファクタというものもあって、これは隣り合ったセクタ番号の差である。

HuBASIC →  $2 - 1 = 1$

X1 CP/M →  $14 - 1 = 13$

MZ CP/M →  $3 - 1 = 2$

となる。MZ CP/Mの場合は15→2, 16→1の所で乱れている点に注意。ま、どーでもいいけどね。

この先は少々その筋の話になるのだが、その筋のディスクにその筋すると、その筋のセクタ番号があったりする。たとえば、第0セクタとか、第17セクタである。さらには第1セクタが26個あったりなどの凶器攻撃もある。ここらへんのことは自由研究としてにおいて、私は次にリードトラックとライトトラックを説明するのである。

X1/X1 turboでリードトラックを実行すると、どちらでもLOST DATAエラーが起きるが、turboではデータが使いものになる理由を説明する。これはディスクのインターフェイスの性能に関係するのである。だからturboにCZ-500FやCZ-800Fを接続してリードトラックを実行しても、使いものになるデータが読めるのである(所々でビットずれは起



こすか、それはいかなる場合でも起こることである)。これは、どうやら VFO というクロックビットとデータビットを分離する回路の性能に関係するらしい。turbo はその回路の性能がいま一なので、取り出したデータは使えるが LOST DATA エラーが発生し、turbo 以外は、いま三なのでエラーが起きるだけではなく、データも狂いまくるわけである。しかし、リードトラック以外では問題が起きないので、あんまり強く文句を言うわけにはいかない——が、私は文句を言うぞっ！

### 「文句、文句、文句、文句、文句」

では、ライトトラックを説明する。そうすればリードトラックも自動的に理解できる。ライトトラックの機能は、言うまでもなく、ディスクに物理フォーマットを書き込むことにある。物理フォーマットは、1トラック(片面分)当たり約 6.25 K バイト(6400 バイト)のデータを、ライトトラックコマンドを使ってディスクに書き込むことにより指定される。その 6.25 K バイトの内訳は、「はいっ、ここからここまでは ID フィールドですよ。そいで、ここからの 256 バイト分はデータです。第 1 セクタの次には第 2 セクタが続きます」などということの指定である。では早速であるが、「ISO タイプ 5.25 インチフォーマット」に使う 6.25 K バイト分のデータを表 9-13 に示す。この表は、参考文献 10 に大いにお世話になっている。

さて、実はこいつはなかなかの喰わせものなのである。よって説明する。

まず、表 9-13 から分かるようにフォーマットは三つの部分に分かれている。GAP1 と各セクタのフォーマット(複数個ある)と GAP4 である。つまり、1トラックのフォーマットは GAP1 で始まり、最後に GAP4 で終わるということである。しかし主役は、その間に挟まれた「セクタのフォーマット」である。1トラック(片面)に、256 バイトの容量のセクタが 16 個ある普通のフォーマットでは、

GAP1, 各セクタ×16, GAP4

となっているわけである。

では、その普通のフォーマットについて、インデックスホールの位置から順に説明する。

#### ● GAP1

インデックスホールの位置から、 $4E_H$ の相当するデータが 32 バイト分である。このほかにも GAP2, 3, 4 があるが、いずれもフロッピーディスクドライブのばらつき(回転速度のズレなど)に対処するためのものである。

#### ●最初のセクタ(第 1 セクタとは限らない)

ここから表 9-13 の下の部分へ行く。

セクタは次の部分からなる。

#### Sync

$00_H$ が 12 個書かれている。これは同期を取るためのもの。

#### AM1

ID ADDRESS MARK である。これは 3 個の  $A1_H$ (実はちょっと違う)と 1 個の  $FE_H$ からなる。 $A1_H$ は、普通のデータではなく、ミッシングクロック(Missing Clock)を含んで



表 9-13 ISO タイプ 5.25 インチフォーマット

1 セクタ当たりの容量	GAP1	各セクタのフォーマット	GAP4
256 バイト	4E <sub>H</sub> ×32	①×16	4E <sub>H</sub> ×266
512 バイト		②×9	4E <sub>H</sub> ×296
1024 バイト		③×5	4E <sub>H</sub> ×208
実際のデータ(MB8877 用)	同 上	下を見よ	同 上

(第 t シリンダ, 第 s サイド, 第 n セクタ用のフォーマット: 実際はこれが複数個並んでいる)

	容 量	Sync	AM1		ID			CRC	GAP 2	Sync	AM 2		DATA	CRC	GAP 3
①	256	00 <sub>H</sub> ×12	(A1 <sub>H</sub> )×3	FE <sub>H</sub> ×1	t	s	n	1	(?)×2	4E <sub>H</sub> ×22	00 <sub>H</sub> ×12	(A1 <sub>H</sub> )×3	FB <sub>H</sub> 256 バイト	(?)×2	4E <sub>H</sub> ×54
②	512							2					または 512 バイト		4E <sub>H</sub> ×84
③	1024							3					1024 バイト		4E <sub>H</sub> ×116
実際のデータ (MB8877 用)		同 上	F5 <sub>H</sub> ×3	同 上	同 上			F7 <sub>H</sub> ×1	同 上	同 上	F5 <sub>H</sub> ×3	同 上	同 上	F7 <sub>H</sub> ×1	同 上

ID ADDRESS MARK

ID フィールド

DATA MARK もしくは  
DELETED DATA MARK

データフィールド

1 セクタ分のフォーマット

いる。これは英語的な言い方で、実にその筋なので言い直すと、よーするにクロックビットが1個足りないのだ。A1<sub>H</sub>は2進数だと、

10100001<sub>B</sub>

となり、これは MFM 記録方式の規則から、0 と 0 の間にクロックビットが入るので、普通ならば、

1 0 1 0 0 0 0 1  
  ↑ ↑ ↑

の3か所にクロックビットが入るはずである。ところがどっこい、この A1<sub>H</sub>の場合は、真ん中のクロックビットがないのである。だから表 9-13 の下の部分では (A1<sub>H</sub>) のように、カッコでくくってある。これは後で出てくる AM2 でも同じである。(A1<sub>H</sub>) の後の FE<sub>H</sub> は普通の FE<sub>H</sub> である。さて、下の方に「実際のデータ」という欄があるが、ここに「F5<sub>H</sub>」とか書いてある。つまり MB8877 にこのクロックビットの抜けている (A1<sub>H</sub>) を書かせるときは、A1<sub>H</sub> の代わりに F5<sub>H</sub> を送ってやればよいのだ。MB8877 はライトトラックを実行中に、F5<sub>H</sub>、F6<sub>H</sub>、F8<sub>H</sub> などが送られてくると特別扱いするのである。この点については後でまとめてやる。

## ID

ここには、シリンダ番号、サイド番号、セクタ番号、セクタのタイプ(大きさ)などが書かれている。リスト 9-6 の「リードアドレス」で読み出したデータのうちの最初の4バイト分である。



## CRC

この CRC は、AM1 の最後の FE<sub>H</sub> と ID の計 5 バイトに対する CRC である。MB8877 では F7<sub>H</sub> を 1 バイト送っただけで、2 バイトの CRC を勝手に計算して書いてくれる。

## GAP2

GAP1 と同じようなもの。

## Sync

さっき説明した。

## AM2

ここにある 3 個の (A1<sub>H</sub>) は AM1 と同じである。その次に FB<sub>H</sub> が来るのだが、これは先程やったライトデータの a<sub>0</sub> フラグと関係しているのだ。普通は FB<sub>H</sub> (DATA MARK) なのだが、a<sub>0</sub> = 1 でライトデータを行なうと F8<sub>H</sub> (DELETED DATA MARK) になってしまうのである。

## DATA

普通読み書きするのはこの部分である。

## CRC

AM2 の FB<sub>H</sub> (もしくは F8<sub>H</sub>) とデータを併せた部分の CRC である。

## GAP3

GAP1, 2 と同じ。

これで 1 セクタ分が終わりである。

### ● 2～16 番目のセクタ

これは「最初のセクタ」と基本的に同じ形式。ただセクタ番号だけが違う。

### ● GAP4

4E<sub>H</sub> が、インデックスホールまで続いているのである。

以上がトラック (しつこいようだが片面分) のフォーマットである。途中で、「F5<sub>H</sub> を送ると (A1<sub>H</sub>) が書かれる」などと書いたが、そのようなことになる値の一覧表を表 9-14 に示す。表 9-14 ④の中では (C2<sub>H</sub>) とかが出てくるが、これは「IBM タイプフォーマット」の中で使われるものなのである。実は 5.25 インチのフロッピーディスクのフォーマットには、メジャーなものが 2 とおりあるのだ。二つのフォーマットの違いは大きくはなく、表 9-15 を表 9-13 の代わりに使うと「IBM タイプ」になる。見て分かれるとおりに、これは GAP4 を短くして GAP1 を増やし、さらに GAP0 とか INDEX MARK とかを入れてしまったものである。ただし全体のバイト数は変わっていないことに注意していただきたい。二つのフォーマットの違いは、参考文献 10 によると、ISO タイプの方がエラー発生の可能性が減るということらしい。たしかに表を見比べると IBM タイプは GAP を短くして INDEX MARK などというものを入れているのだから、FDC がどちらのフォーマットも読めるのであれば、ゆとりのある ISO タイプの方が有利であろう。ちなみに HuBASIC と S-BASIC (MZ-2000) は ISO タイプ、MZ-2000 の CP/M は大胆にも IBM タイプのようである。



表 9-14 ライトトラック(フォーマット)時のデータの意味

① MFM 動作時

MB8877/76 に送るデータ	ディスクに 書かれるデータ	意 味
00 <sub>H</sub> } F4 <sub>H</sub>	00 <sub>H</sub> } F4 <sub>H</sub>	普通のデータ
F5 <sub>H</sub>	(A1 <sub>H</sub> )	AM1, AM2 の前提(CRC ジェネレータをプリセットする)
F6 <sub>H</sub>	(C2 <sub>H</sub> )	INDEX MARK の前提(CRC ジェネレータをプリセットする)
F7 <sub>H</sub>	CRC	内部で計算された CRC (2 バイト)を書く
F8 <sub>H</sub>	F8 <sub>H</sub>	AM2 の中では DELETED DATA MARK
F9 <sub>H</sub> FA <sub>H</sub>	F9 <sub>H</sub> FA <sub>H</sub>	普通のデータ
FB <sub>H</sub>	FB <sub>H</sub>	AM2 の中では DATA MARK
FC <sub>H</sub>	FC <sub>H</sub>	(C2 <sub>H</sub> )×3 の後で INDEX MARK
FD <sub>H</sub>	FD <sub>H</sub>	普通のデータ
FE <sub>H</sub>	FE <sub>H</sub>	AM1 の中では ID ADDRESS MARK
FF <sub>H</sub>	FF <sub>H</sub>	普通のデータ

② FM 動作時

MB8877/76 に送るデータ	ディスクに書かれるデータ		意 味
	データ パターン	クロック パターン	
00 <sub>H</sub> } F4 <sub>H</sub>	00 <sub>H</sub> } F4 <sub>H</sub>	FF <sub>H</sub>	普通のデータ
F5 <sub>H</sub>	禁 止		
F6 <sub>H</sub>			
F7 <sub>H</sub>	CRC	FF <sub>H</sub>	内部で計算された CRC (2 バイト)を書く
F8 <sub>H</sub>	F8 <sub>H</sub>	C7 <sub>H</sub>	AM2 の中では DELETED DATA MARK (CRC ジェネレータをプリセットする)
F9 <sub>H</sub> FA <sub>H</sub>	F9 <sub>H</sub> FA <sub>H</sub>	C7 <sub>H</sub>	CRC ジェネレータをプリセットする
FB <sub>H</sub>	FB <sub>H</sub>	C7 <sub>H</sub>	DATA MARK (CRC ジェネレータをプリセットする)
FC <sub>H</sub>	FC <sub>H</sub>	D7 <sub>H</sub>	INDEX MARK (CRC ジェネレータをプリセットする)
FD <sub>H</sub>	FD <sub>H</sub>	FF <sub>H</sub>	普通のデータ
FE <sub>H</sub>	FE <sub>H</sub>	C7 <sub>H</sub>	ID ADDRESS MARK (CRC ジェネレータをプリセットする)
FF <sub>H</sub>	FF <sub>H</sub>	FF <sub>H</sub>	普通のデータ

注) FM 方式の場合、普通のデータのクロックパターンは「FF<sub>H</sub>」である。



表 9-15 IBM タイプ 5.25 インチフォーマット

1 セクタ当たりの容量	GAP0	Sync	INDEX MARK		GAP1	各セクタのフォーマット	GAP4
256 バイト	4E <sub>H</sub> ×80	00 <sub>H</sub> ×12	(C2 <sub>H</sub> )×3	FC <sub>H</sub> ×1	4E <sub>H</sub> ×50	①×16	4E <sub>H</sub> ×152
512 バイト						②×9	4E <sub>H</sub> ×132
1024 バイト						③×5	4E <sub>H</sub> ×94
実際のデータ	同 上	同 上	F6 <sub>H</sub> ×3	同 上	同 上	表9-13の下の部分を見よ	同 上

そこで、どどーんとディスクをフォーマットするプログラムがリスト 9-9 である（注釈を見ると turbo BASIC だということが分かるが、実際は turbo BASIC では動かないので注意）。1010～1170 行は例の機械語部分（リスト 9-5 の 110～270 行）をリナンバーしたものである。二、三説明すると、まずこのプログラムは CZ-8FB01 でなければ動かない。それから 1340～1380 行は、USR0 を実行してエラーが起きた場合に 5 回まで試してみるためのものである。このプログラム（機械語部分）だと、どうもタイミングがいまいちらしく、変数 STAT に返ってくるステータスがたまに「2」になってしまう。これは「DATA REQUEST だけ」ということなのだ。「BUSY」にはなっていないのである。困ったものだ。しかし、たまにだけだから許すことにしたのである。次に 1860～1900 行であるが、これはシリンダ番号、サイド番号、セクタ番号、セクタ容量、のメモリへの書き込みを行なっている。データに対応するアドレスはサブルーチン「MF-SUB」によって配列 VAR (～) に入れられてあるのだ。RUN すると、ドライブ「1:」に入っているディスクを、のんびりとフォーマットする。各トラック（片面）ごとにステータスを表示し、0 でなかったら数度（5 回まで）リトライするようになっている。

リスト 9-9 ディスクフォーマットプログラム(turbo BASIC では動かない)

```

1000 CLEAR &HC000
1010 MEM$(&HDF00,16)=HEXCHR$("D5 1A 13 D5 21 1A DF 16 00 87 5F 19 5E 23 56 EB")
1020 MEM$(&HDF10,16)=HEXCHR$("D1 F3 CD 19 DF FB D1 12 C9 E9 32 DF 3C DF 55 DF")
1030 MEM$(&HDF20,16)=HEXCHR$("55 DF 55 DF 5F DF 84 DF 5F DF 5F DF 84 DF BC DF")
1040 MEM$(&HDF30,16)=HEXCHR$("C6 DF 1A 01 F8 0F ED 79 CD D2 DF C9 EB 7E 23 56")
1050 MEM$(&HDF40,16)=HEXCHR$("23 5E 01 F9 0F ED 59 01 FB 0F ED 51 01 F8 0F ED")
1060 MEM$(&HDF50,16)=HEXCHR$("79 CD D2 DF C9 1A 01 F8 0F ED 79 CD D2 DF C9 CD")
1070 MEM$(&HDF60,16)=HEXCHR$("EA DF 01 F8 0F D9 01 FB 0F 2A FE DF D9 ED 79 CD")
1080 MEM$(&HDF70,16)=HEXCHR$("E4 DF ED 78 0F 30 0B 0F 30 F8 D9 ED 78 77 23 D9")
1090 MEM$(&HDF80,16)=HEXCHR$("18 F0 07 C9 CD EA DF 01 F8 0F D9 01 FB 0F 2A FE")
1100 MEM$(&HDF90,16)=HEXCHR$("DF D9 1E 00 16 02 ED 79 ED 78 A2 C2 A4 DF 1D C2")
1110 MEM$(&HDFA0,16)=HEXCHR$("98 DF 18 06 D9 7E ED 79 23 D9 ED 78 0F 30 0B 0F")
1120 MEM$(&HDFB0,16)=HEXCHR$("30 F8 D9 7E ED 79 23 D9 18 F0 07 C9 1A 01 F8 0F")
1130 MEM$(&HDFC0,16)=HEXCHR$("ED 79 CD D2 DF C9 1A 01 FC 0F ED 79 B7 F0 CD D2")
1140 MEM$(&HDFD0,16)=HEXCHR$("DF C9 C5 06 20 10 FE 01 F8 0F ED 78 4F E6 81 20")
1150 MEM$(&HDFE0,16)=HEXCHR$("F6 79 C1 C9 3E 07 3D 20 FD C9 EB 7E 23 56 01 FA")
1160 MEM$(&HDFF0,16)=HEXCHR$("0F ED 51 F5 CD D2 DF F1 C9 00 00 00 00 00 00 E0")
1170 DEFUSR0=&HDF00
1180 DEFINT A-Z:DIM VAR(40),SQ(40)
1190 DN=1 : 'ドライブ番号
1200 AD=&HE000 : 'フォーマット用データのアドレス
1210 PRINT"MAKING DATA":GOSUB"MAKE-FORMAT":'フォーマット用データを作る
1220 '
1230 DUMMY=INP(&HFFF) : '2 D, 2 D D モードにする
1240 OUT &HFFC,&H80+DN:GOSUB"WNBSY" : 'モーター ON
1250 OUT &HFF8,&H0 :GOSUB"WNBSY" : 'リストア
1260 OLDTR=0
1270 RESTORE"TYPE":READ SECTORTYPE
1280 '
1290 FOR CYLINDER=0 TO 39
1300 GOSUB"SEEK":OLDTR=CYLINDER:'SEEK
1310 FOR SIDE=0 TO 1
1320 OUT &HFFC,&H80 OR DN OR SIDE*&H10 : '裏表の設定

```



```

1330 GOSUB"SET-FORMAT"
1340 TRY=0
1350 IF TRY>5 THEN BEEP:PRINT:PRINT"ERROR":GOTO 1430
1360 GOSUB"WTR" :WRITE TRACK
1370 PRINT STAT;";": :PRINT STATUS
1380 IF STAT<>0 THEN TRY=TRY+1:GOTO1350 :RETRY
1390 NEXT
1400 NEXT
1410 'DMA$=HEXCHR$("83"):GOSUB"SETDMA"
1420 PAUSE 1
1430 OUT &HFFC,&H3 AND DN:'モーターOFF
1440 DUMMY=INP(&HFFF) :2D,2DDモードにする
1450 END
1460 '
1470 LABEL"SEEK"
1480 OUT &HFFB,CYLINDER :データレジスタ
1490 OUT &HFF9,OLDTR :トラックレジスタ
1500 OUT &HFF8,&H10 :シークコマンド
1510 GOSUB"WNBSY"
1520 RETURN
1530 '
1540 LABEL"WNBSY"
1550 CT=0
1560 IF CT>1000 THEN PRINT"DISK?":GOTO 1410
1570 IF INP(&HFF8) AND &H81 THEN CT=CT+1:GOTO 1560
1580 RETURN
1590 '
1600 LABEL"SETDMA"
1610 FORI=1TOLEN(DMA$):OUT &H1F80,ASC(MID$(DMA$,I,1)):NEXT
1620 RETURN
1630 '
1640 LABEL"WTR"
1650 D2$=USR0(CHR$(9,&HF0,1)):STAT=ASC(D2$):RETURN
1660 DMA$=HEXCHR$("83 79 00 C0 AF 28 14 28 8D FB 0F 92 CF 05 CF 87")
1670 GOSUB"SETDMA"
1680 OUT &HFF8,&HF0 :ライトトラックコマンド
1690 S=INP(&HFF8):IF S AND 1 THEN 1690
1700 STAT=S:RETURN
1710 '
1720 LABEL"MAKE-FORMAT"
1730 VAR(0)=0:'CLEAR COUNTERS
1740 RESTORE"GAP1":GOSUB"MF-SUB" :MAKE GAP1
1750 RESTORE"TOTAL SECTOR":READ TSEC :HOW MANY SECTORS ?
1760 FOR SECT=1 TO TSEC
1770 RESTORE"SECTOR":GOSUB"MF-SUB" :MAKE SECTOR FORMAT
1780 NEXT
1790 RESTORE"GAP4":GOSUB"MF-SUB" :MAKE GAP4
1800 RESTORE"SQUE"
1810 FOR SECT=1 TO TSEC
1820 READ SQ(SECT)
1830 NEXT
1840 RETURN
1850 '
1860 LABEL"SET-FORMAT":SET ID FIELD
1870 FOR P=1 TO VAR(0)
1880 MEM$(VAR(P),4)=CHR$(CYLINDER,SIDE,SQ(P),SECTORTYPE)
1890 NEXT
1900 RETURN
1910 '
1920 LABEL"MF-SUB"
1930 READ C:IF C=0 THEN RETURN:' n Byte,DATA ノケイシキ
1940 READ D$:IF D$<>"!" THEN D=VAL(D$):GOTO 1970
1950 '!...MEANS "VARIABLE DATA"
1960 VAR(0)=VAR(0)+1:VAR(VAR(0))=AD:D=99:'INC COUNTER, STORE ADDRESS
1970 IF C<256 THEN MEM$(AD,C)=STRING$(C,D):AD=AD+C:GOTO1930
1980 MEM$(AD,255)=STRING$(255,D):AD=AD+255:C=C-255:GOTO1970
1990 '
2000 LABEL"GAP1"
2010 DATA 32,&H4E,0
2020 LABEL"SECTOR"
2030 DATA 12,0,3,&HF5,1,&HFE,4,!
2040 DATA 1,&HF7,22,&H4E,12,0,3,&HF5,1,&HFB,256,&HE5,1,&HF7,54,&H4E,0
2050 LABEL"GAP4" :
2060 DATA 266,&H4E,0 :'-!!! !!! !!!
2070 LABEL"TOTAL SECTOR"
2080 DATA 16 :'-!!!
2090 LABEL"SQUE"
2100 DATA 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16
2110 LABEL"TYPE"
2120 DATA 1 :'-!!! 1 SECTOR 256 Byte つうことでんがな

```



さて、このプログラムは実にその筋なので、2000 行から後をリスト 9-10 と替えると、なんと 1 セクタ 512 バイトのフォーマットとなるのである。さらにはリスト 9-11 に替えると、1 セクタが 1024 バイトになってしまうのだ。ディスク 1 枚で 400K バイトだぞ。さらに追い撃ちだっ。3 インチドライブや X1 turbo などのドライブ (CZ-800F/801F 以外) だとヘッドは第 42 シリンダまで動くから、1290 行の「39」を「42」にするとディスク 1 枚で 430 K バイトだだだだだっ！ 持ってけドロボーなのである。これは「オーバートラック」と呼ばれるヒネリ技である。もちろん第 40 シリンダ以降の書き込み、および書き込まれたデータには、シャープ株式会社は何の保証もしない「その筋トラック」であるから覚悟して使うように。

リスト 9-10 ディスクフォーマットプログラム 1 セクタ = 512 バイト用変更点

```

2000 LABEL "GAP1"
2010 DATA 32,&H4E,0
2020 LABEL "SECTOR"
2030 DATA 12,0,3,&HF5,1,&HFE,4,!
2040 DATA 1,&HF7,22,&H4E,12,0,3,&HF5,1,&HFB,512,&HE5,1,&HF7,84,&H4E,0
2050 LABEL "GAP4" :'^
2060 DATA 296,&H4E,0 : '<-!!! !!! !!!
2070 LABEL "TOTAL SECTOR"
2080 DATA 9 : '<-!!!
2090 LABEL "SQUE"
2100 DATA 1,2,3,4,5,6,7,8,9
2110 LABEL "TYPE"
2120 DATA 2 : '<-!!! 1 SECTOR 512 Byte つうことでんがな

```

リスト 9-11 ディスクフォーマットプログラム 1 セクタ = 1024 バイト用変更点

```

2000 LABEL "GAP1"
2010 DATA 32,&H4E,0
2020 LABEL "SECTOR"
2030 DATA 12,0,3,&HF5,1,&HFE,4,!
2040 DATA 1,&HF7,22,&H4E,12,0,3,&HF5,1,&HFB,1024,&HE5,1,&HF7,116,&H4E,0
2050 LABEL "GAP4" :'^
2060 DATA 208,&H4E,0 : '<-!!! !!! !!!
2070 LABEL "TOTAL SECTOR"
2080 DATA 5 : '<-!!!
2090 LABEL "SQUE"
2100 DATA 1,2,3,4,5
2110 LABEL "TYPE"
2120 DATA 3 : '<-!!! 1 SECTOR 1024 Byte つうことでんがな

```

なお、このプログラムには仕掛けがしてあるので、turbo の場合は少々変更すると DMA を使ったディスクフォーマットルーチンになる。それについてはまた後程。

次にリスト 9-12, 9-13 である。これは G-RAM → ディスク間の直接読み書きルーチンなのだ。リスト 9-12 では、書き込みルーチンがなかなか味わい深い。この部分では、DATA REQUEST 信号に素早く応答できるように A' (裏レジスタ) に、次に書き込むべきデータを用意してあるのだ。タイミングが難しいケースにおける、いわゆる一つの解決例になっている。



リスト 9-12 G-RAM ↔ ディスク直接 R/W プログラム

		.Z80	
		.PHASE	0DD00H
0FF8		;	
0FF8		CR EQU	0FF8H ;COMMAND REG.
0FF9		STR EQU	0FF8H ;STAT. REG.
0FFA		TR EQU	0FF9H ;TRACK REG.
0FFB		SCR EQU	0FFAH ;SECTOR REG.
0FFC		DR EQU	0FFBH ;DATA REG.
		MSDR EQU	0FFCH ;MOTOR,SIDE,DRIVE# SELEC
T			
DD00	D5	;	
DD01	DD E1	START: PUSH	DE
DD03	DD 66 01	POP	IX ;IX=DE
DD06	DD 6E 02	LD	H,(IX+1) ;H=COMMAND
DD09	16 FB	LD	L,(IX+2) ;L=SECTOR#
DD0B	1E F8	LD	D,LOW(DR)
DD0D	D9	LD	E,LOW(STR)
DD0E	DD 4E 04	EXX	
DD11	DD 46 05	LD	C,(IX+4)
DD14	ED 78	LD	B,(IX+5) ;BUFF ADD.
DD16	08	IN	A,(C) ;GET FIRST 1 Byte
DD17	D9	EX	AF,AF' ;A'=FIRST DATA
DD18	DD 7E 00	EXX	
DD1B	F3	LD	A,(IX+0) ;GET 'R' or 'W'
		DI	
DD1C	FE 52	;	
DD1E	28 0A	CP	'R'
		JR	Z,READD
DD20	FE 57	;	
DD22	28 2C	CP	'W'
		JR	Z,WRITD
DD24	DD 36 00 FF	;	
DD28	FB	LD	(IX+0),0FFH ;SET ERROR
DD29	C9	EI	
		RET	
DD2A		;	
DD2A	CD DD77	READD:	
		RAGAIN: CALL	SETFDC ;SET SCT#,COM.
DD2D	ED 78	;	
DD2F	0F	RED1: IN	A,(C) ;GET STATUS
DD30	30 0E	RRCA	;CHECK BUSY
DD32	0F	JR	NC,RED2 ;END READ
DD33	30 F8	RRCA	;CHECK DATA REQ.
		JR	NC,RED1 ;NO REQ.
DD35	4A	;	
DD36	ED 78	LD	C,D ;BC=DR
DD38	D9	IN	A,(C) ;READ DATA
DD39	ED 79	EXX	
DD3B	03	OUT	(C),A ;STORE DATA
DD3C	D9	INC	BC ;NEXT ADD.
DD3D	4B	EXX	
DD3E	18 ED	LD	C,E ;BC=STR
		JR	RED1
DD40	07	;	
DD41	DD 77 00	RED2: RLCA	;BACK STAT
DD44	E6 9F	LD	(IX+0),A ;STORE STAT
DD46	20 06	AND	9FH ;CHECK ERROR
DD48	2C	JR	NZ,RED3 ;QUIT READ
DD49	DD 35 03	INC	L ;IX+2=SCTNO
DD4C	20 DC	DEC	(IX+3) ;IX+3=COUNT
DD4E	FB	JR	NZ,RAGAIN
DD4F	C9	RED3: EI	
		RET	
DD50		;	
DD50	CD DD77	WRITD:	
		WAGAIN: CALL	SETFDC ;SET SCT#,COM.
DD53	ED 78	;	
DD55	0F	WRT1: IN	A,(C)
DD56	30 10	RRCA	;CHECK BUSY
DD58	0F	JR	NC,WRT2 ;END WRITE
DD59	30 F8	RRCA	;CHECK DATA REQ.
		JR	NC,WRT1 ;NO REQ.
DD5B	4A	;	
DD5C	08	LD	C,D ;BC=DR
DD5D	ED 79	EX	AF,AF' ;A'=DATA
DD5F	D9	OUT	(C),A ;WRITE DATA
DD60	03	EXX	
DD61	ED 78	INC	BC ;INC ADD.
		IN	A,(C) ;GET NEXT DATA



```

DD63      08                      EX      AF,AF'  ;A'=NEXT DATA
DD64      D9                      EXX
DD65      4B                      LD      C,E      ;BC=DR
DD66      18 EB                    JR      WRT1

;
DD68      07                      WRT2:  RLCA      ;BACK STAT
DD69      DD 77 00                LD      (IX+0),A  ;STORE STAT
DD6C      B7                      OR      A      ;CHECK ERROR
DD6D      20 06                    JR      NZ,WRT3 ;ERROR THEN END
DD6F      2C                      INC      L      ;IX+2=SCTNO
DD70      DD 35 03                DEC      (IX+3) ;IX+3=COUNT
DD73      20 DB                    JR      NZ,WAGAIN
DD75      FB                      WRT3:  EI
DD76      C9                      RET

;
DD77      7D                      SETFDC: LD      A,L      ;IX+2=SCTNO
DD78      01 0FFA                LD      BC,SCR
DD7B      ED 79                    OUT     (C),A
DD7D      01 0FF8                LD      BC,CR   ;CR=STR
DD80      ED 61                    OUT     (C),H
DD82      3E 07                    LD      A,7
DD84      3D                      WAIT:  DEC      A
DD85      20 FD                    JR      NZ,WAIT
DD87      C9                      RET

;
END

```

### リスト 9-13 G-RAM←ディスク直接転送プログラム

```

270 '100 - 270 キーボード ハ リスト9-5 ノモノヲ ツカウノデアル
280 CLEAR &HDD00
290 MEMS(&HDD00,16)=HEXCHRS("D5 DD E1 DD 66 01 DD 6E 02 16 FB 1E F8 D9 DD 4E")
300 MEMS(&HDD10,16)=HEXCHRS("04 DD 46 05 ED 78 08 D9 DD 7E 00 F3 FE 52 28 0A")
310 MEMS(&HDD20,16)=HEXCHRS("FE 57 28 2C DD 36 00 FF FB C9 CD 77 DD ED 78 0F")
320 MEMS(&HDD30,16)=HEXCHRS("30 0E 0F 30 F8 4A ED 78 D9 ED 79 03 D9 4B 18 ED")
330 MEMS(&HDD40,16)=HEXCHRS("07 DD 77 00 E6 9F 20 06 2C DD 35 03 20 DC FB C9")
340 MEMS(&HDD50,16)=HEXCHRS("CD 77 DD ED 78 0F 30 10 0F 30 F8 4A 08 ED 79 D9")
350 MEMS(&HDD60,16)=HEXCHRS("03 ED 78 08 D9 4B 18 EB 07 DD 77 00 B7 20 06 2C")
360 MEMS(&HDD70,16)=HEXCHRS("DD 35 03 20 DB FB C9 7D 01 FA 0F ED 79 01 F8 0F")
370 MEMS(&HDD80,16)=HEXCHRS("ED 61 3E 07 3D 20 FD C9 00 00 00 00 00 00 00")
380 DEFUSR2=&HDD00
390 '
400 GOSUB"GSMPL"
410 DS=USR0(CHR$(11,&H81)) : 'MOTOR ON
420 DS=USR0(CHR$(0,&H2)) : 'RESTORE
430 TR0=0:FTR=0:FSCT=1:COMS="W"+CHR$(&HA0):HI=&H40
440 SCT=256:SPT=16
450 GOSUB"R/W"
460 '
470 BEEP:CLS4
480 DS=USR0(CHR$(0,&H2)) : 'RESTORE
490 TR0=0:FTR=0:FSCT=1:COMS="R"+CHR$(&H80):HI=&H40
500 SCT=256:SPT=16
510 GOSUB"R/W"
520 DS=USR0(CHR$(11,&H1)) : 'MOTOR OFF
530 END
540 '
550 LABEL"R/W":'GIVE ME TR0,FTR,FSCT,COMS,HI,SCT,SPT
560 TR=FTR:SABA=FSCT:PLUS=SCT*SPT/256
570 FOR SIDE=0 TO 1
580 DS=USR0(CHR$(11,&H81+SIDE*16)) : 'MOTOR ON
590 DS=USR0(CHR$(1,&H1E,TR,TR0)):TR0=TR : 'SEEK
600 IF HI+PLUS>=&H100 THEN P=(&H100-HI)/(SCT/256) ELSE P=SPT
610 IF SABA<>1 THEN P=P-SABA
620 DS=USR2(COMS+CHR$(SABA,P,0,HI)):SABA=1
630 PRINT ASC(D$),SABA,P,HEX$(HI)
640 HI=HI+P*(SCT/256):IF HI>=&H100 THEN RETURN
650 NEXT
660 TR=TR+1:GOTO570
670 '
680 LABEL"GSMPL"
690 CLS4:INIT
700 FOR I=0 TO 10:X1=INT(RND(1)*640):Y1=INT(RND(1)*200):C=INT(RND(1)*7)+1
710 LINE(X,Y)-(X1,Y1),XOR,C:X=X1:Y=Y1
720 NEXT:RETURN

```



リスト 9-13 が応用例だが、G-RAM → FD の機械語ルーチン、USR2 の呼び方は、  
USR2 ( { "R" } + CHR\$ ( コマンド, 先頭セクタ, セクタ数, アドレス下位, アドレス  
上位 ) )

である。アドレスとはデータの始まる G-RAM のアドレスである。このアドレスは、グラフィック全画面のセーブ/ロードなどでは頻繁に変化するものなので、POKE 文を使いたくなかったのだ。その見返りとして暴走しやすくなっているから、充分に楽しんでいただきたい。プログラムでは 680 行からのサブルーチンででたらめな折れ線を引き、その後フロッピーディスクにセーブ、画面クリア、G-RAM へのロードを行なっている。興味があったので、シークも含めてすべてを機械語で組んでみたところ(載せないよーだ)、理論的限界に達していることが判明した。すなわち、1 周分の 16 セクタを読み書きするには 0.2 秒が必要だから、全画面をセーブ/ロードするにはシーク時間を入れなくても、どうしても 2.4 秒以上かかるのである。大体において、その程度の時間で読めているから、それ以上のことをするにはリスト 9-10, 9-11 などを使って 1 トラック当たりの容量を増やすなどしなければならない。1 セクタ 1024 バイトのフォーマットならば、5 シリンダ (50K バイト) で済むから、全画面ロード/セーブはシーク時間を入れずに 2 秒が限界になる(しかもシークが 1 回少なくて済む)。FDC の楽しみはこのようにして深く、広くなっていくのであった。

## TYPE IV

X1 の場合はコマンドの打ち切り機能だけである。

## そしてDMAである

ここから先では DMA が必要なので、turbo/turboZ ユーザーだけに意味がある。

さて、2D, 2DD のディスクを扱うのならば DMA がなくとも子は育つのだが、2HD となるとそうはいかない。データの転送速度が 2 倍になるので、CPU でシャコシャコとデータを送っていたならば間に合わなくなってしまうのである。そこで DMA を使ったディスクアクセスへと向かうわけである。ちなみに、リスト 9-4 の DMA を使わずにディスクアクセスするプログラムで試したところ、やはりしっかりと「LOST DATA エラー」が起きてしまった。そのよーなわけであるから、やはり DMA はただ者ではないのである。

Z は別だが、turbo ユーザーで 2HD を持っている人はあまりいないであろう。しかしご安心。DMA を使ったディスクアクセスは、2HD ばかりではないのである。すなわち、

**2D, 2DD でもできるのだ。**

その場合のメリットは、

- 1) プログラムが短くなる
- 2) 恐ろしいことに、**オール BASIC** で書けてしまう



という2点である。念のために言っておくと、当たり前のことだが、2D、2DDではDMAを使っても使わなくても、ディスクの回転速度は変わらないのであるから、基本的にリード/ライトの速さは変わらない。

ところでCZ-520FやturboIII、Zの内蔵ドライブなどの2D、2DD、2HD兼用タイプのドライブを使ううえで大事な注意点がある。

それは、それらのドライブは2Dの書き込みはできるが、書き込みをしたならば他の2D専用のドライブでの読み込みは保証されないということである(2DDは問題ない)。

具体的に言うと、たとえばスイッチを切り換えて2Dモードにする。その状態で、ディスク(もちろん2Dのフォーマットになっている)に何かを書いたとする。そのディスクをそのタイプのドライブで読む分にはなんの問題もないのだが、他の2D専用のドライブ(X1G、turbo、turboIIの内蔵ドライブ、CZ-800F、801F、502Fなど)に差し替えたとなると、リードエラーが起きる可能性があるのだ。もの好きな私であるから早速試したが、turboの内蔵ドライブでは34番トラックあたりまではよかったのだが35番トラックでリードエラーが起きてしまった。ところがどっこいCZ-800Fではすんなりと読めてしまったのである。そのよーな状況であるから、注意していただきたい。なお、この点は他機種ドライブにおいても状況は同じだそうである。

なお、知っている人は知っていると思うが、2HDではディスクの回転速度が毎分360回(=毎秒6回)であり、2D、2DDでは毎分300回(=毎秒5回)である。すなわち2D、2DD、2HD兼用タイプのドライブは、動作モードによって回転速度が自動的に変わるのである。

## 脱線である

ここで一言断っておく。今発見したところなのだが、turbo、turbo IIなどでは、ノーマルの2D専用ドライブで(つまり2HDを扱えないドライブで)5インチの単密度(FM方式)が使えるようなのである(つまり5インチの2Sというフォーマット)。私は今まで2Sも1Sも見ることがないし、読者もほとんどの方が見たことがないであろう(Apple IIは1Sに近いそうだが、ちょっと違う)。そのよーなわけで、以後の文中ではこのフォーマットは無視する。なにせすでに死に絶えてしまったフォーマットであるから、使い道はないのである。製造元のシャープでも2Sについては何も保証しないだろうと思われる。

## 本題である

X1の5インチ2HDの物理フォーマットにはX1フォーマットと標準フォーマットの2種類がある。X1フォーマットとは全部MFMで、77シリンダ、2サイド、各サイドには77トラックあり、1トラックには26セクタあり、セクタの容量は256バイトの(ああ面倒臭い)フォーマットである。標準フォーマットとは、IBMフォーマット(8インチ)と同じフォーマットで、第0シリンダのサイド0が単密度になっているものである。



というわけで、ここでは X1 フォーマットを基本として話を進める。まずはリスト 9-14 である。これはリスト 9-5 に相当するものである。つまり、FDC の MB8877/8876 のコマンドをすべて実行できるようにしてある。BASIC は turbo BASIC (CZ-8FB02) を使っていただきたい。使い方は四つの変数、TR, OLDTR, SCT, CMD に必要な値を入れて GOSUB するのである。実行に当たっては、打ち込みミスがあると悲惨なことになるので、壊れては困るファイルが入っているディスクは、すべてのドライブから抜いておくこと。また一度このプログラムを走らせた後に、LOAD, SAVE を行なうときには、まずは FILES を実行していただきたい。もしディスク周りのモードがおかしくなっていたなら、変なものが表示されるから、そのときはディスクアクセスは**すべきではない**。そして、素直に IPL をかけて、バグ捜しとなる。DMA がその筋すると電源を落とさなければならない場合もあるので心得ていただきたい。なお、1020 行と 1150 行に「CMD=-1」という理不尽な式があるが、これはただの縁起ものと考えていただきたい。パッファ用のデータ領域は C000<sub>H</sub>〜を使っている。リードトラックを実行すると、E8B0<sub>H</sub>あたりまで使うことになるので、まだ 4K バイトほど余裕がある。では 1190 行から始まるサブルーチンを順に解説する。

- “SET HD”

2HD の MFM モードにする。正確には、1200 行で 2HD モードにし、1210 行で MFM モードにしている。2HD とはいっても、MFM (倍密度) と FM (単密度) の 2 種類があるのだ。

- “SET LD”

2D, 2DD モードにする。

- “SET FM”

2HD の FM モード (単密度) にする。

- “MOTOR”

ドライブのモーターの ON/OFF, サイドの選択をする。

- “RESTORE”

第 0 トラックヘシークする (ヘッドを移動する)。

- “SEEK”

ヘッドを移動する。始まりのトラック番号は OLDTR, 行き先のトラック番号は TR である。

- “STEP”

直前に動いた方向に、もう 1 トラック分シークする。

- “STEP IN”

内側 (トラック番号が増える方向) に 1 トラック分シークする。

- “STEP OUT”

外側 (トラック番号が減る方向) に 1 トラック分シークする。

- “READ DATA”

SCT で指定された番号のセクタを C000<sub>H</sub>から読み出す。

さて、ここで初めて DMA の登場である。DMA に対するコントロール内容は 1700 行に



あるとおり。すなわち、

83<sub>H</sub>=WR6 で DMA 停止の意味。

7D<sub>H</sub>=WR0 で動作は転送で、方向はポート A →ポート B を指定。次にポート A の開始アドレスとブロック長が来る。

FB<sub>H</sub>, 0F<sub>H</sub>=FDC のデータレジスタの I/O アドレス。

FF<sub>H</sub>, 00<sub>H</sub>=ブロック長(DMA の都合により-1 しておく)。1 セクタが 256 バイトだからこうなる。

2C<sub>H</sub>=WR1 でポート A はアドレスが固定で、I/O であることを指定。

10<sub>H</sub>=WR2 でポート B はアドレスがインクリメント (増加) でメモリであることを指定。

80<sub>H</sub>=WR3 で特に何もしてない。

8D<sub>H</sub>=WR4 でバイトモードを指定。次にポート B の開始アドレスが来る。

00<sub>H</sub>, C0<sub>H</sub>=メモリのデータ領域 (バッファ) のアドレス。

92<sub>H</sub>=WR5 で CE/WAIT をマルチプレクス、またレディの極性は Low。

CF<sub>H</sub>=WR6 でロードコマンド。

87<sub>H</sub>=WR6 で DMA イネーブル。

以上のように DMA を設定してやると、DMA は FDC からのレディ信号を待つ状態になる。そこで FDC にリードコマンドを送ってやると、その後 FDC と DMA の間で勝手にデータの転送をしてくれる。DMA はバイトモードであるから、転送する間を縫って CPU が動作している。データリードが終わったかどうかは、CPU が FDC のステータスレジスタを見張って、BUSY でなくなったかで判断する。終わったならば DMA に 83<sub>H</sub>を送って動作を停止させておく。

#### ●“WRITE DATA”

SCT で指定された番号のセクタに C000<sub>H</sub>からのデータを書き込む。

DMA に対するコントロール内容は、

83<sub>H</sub>=WR6 で DMA 停止の意味。

79<sub>H</sub>=WR0 で動作は転送で、方向はポート B →ポート A を指定(後でポート A →ポート B にひっくり返す)。次にポート A の開始アドレスとブロック長が来る。

00<sub>H</sub>, C0<sub>H</sub>=データ領域のアドレス。

FF<sub>H</sub>, 00<sub>H</sub>=ブロック長(DMA の都合により-1 しておく)。1 セクタが 256 バイトだからこうなる。

14<sub>H</sub>=WR1 でポート A はアドレスがインクリメント (増加) で、メモリであることを指定。

28<sub>H</sub>=WR2 でポート B はアドレスが固定で I/O であることを指定。

80<sub>H</sub>=WR3 で特に何もしてない。

8D<sub>H</sub>=WR4 でバイトモードを指定。次にポート B の開始アドレスが来る。

FB<sub>H</sub>, 0F<sub>H</sub>=FDC のデータレジスタの I/O アドレス。

92<sub>H</sub>=WR5 で CE/WAIT をマルチプレクス、またレディの極性は Low。

CF<sub>H</sub>=WR6 で、ロードコマンド。これにより今のところ 2 バイト目の 79<sub>H</sub>でソース側



に指定されているポート B に開始アドレス (0FFB<sub>H</sub>) がロードされる。これを怠ると正常に動作しない。

05<sub>H</sub>=WR0 でポート A →ポート B に転送方向を指定し直している。

CF<sub>H</sub>=WR6 でもう一度ロードコマンド。

87<sub>H</sub>=WR6 で DMA イネーブル。

以上の設定の後に FDC にデータライトコマンドを送ってやると、DMA と FDC が勝手に書き込みをしてくれる。そのほかは “READ DATA” と同じ。

- “READ ID”

ブロック長が 6 バイトになったこと以外は “READ DATA” と同じ (リスト中では - 1 して 05<sub>H</sub>, 00<sub>H</sub>となっている)。

- “READ TRACK”

ブロック長が 28B0<sub>H</sub> バイトになったこと以外は “READ DATA” と同じ。

- “WRITE TRACK”

ブロック長が 28B0<sub>H</sub> バイトになったこと以外は “WRITE DATA” と同じ。

- “FORCE INT”

FDC にコマンドを送るだけ。

このよーにして DMA を使ったディスクアクセスはオール BASIC してしまうのであった。リスト 9-14 では CZ-520F をドライブ 2, 3 とした場合を考えている。1020 行を GOSUB “SET LD” にし、1030 行などで、“MOTOR” に与えるコマンド (CMD) を &H81 などにとすると内蔵のドライブ 1 に DMA を使ってアクセスできる。

リスト 9-14 DMA でディスクアクセス

```
1000 CLEAR &HC000
1010 'TR,OLDTR,SCT,CMD
1020          CMD=-1 :GOSUB"SET HD"          :GOSUB"ER?"
1030          CMD=&H82:GOSUB"MOTOR"          :GOSUB"ER?"
1040          CMD=&H0 :GOSUB"RESTORE"         :GOSUB"ER?"
1050 'TR=1 :OLDTR=0 :CMD=&H1C:GOSUB"SEEK"     :GOSUB"ER?"
1060 '          CMD=&H38:GOSUB"STEP"          :GOSUB"ER?"
1070 '          CMD=&H58:GOSUB"STEP IN"       :GOSUB"ER?"
1080 '          CMD=&H78:GOSUB"STEP OUT"      :GOSUB"ER?"
1090 SCT=1    :CMD=&H80:GOSUB"READ DATA"     :GOSUB"ER?"
1100 SCT=1    :CMD=&HA0:GOSUB"WRITE DATA"    :GOSUB"ER?"
1110 '          CMD=&HC0:GOSUB"READ ID"       :GOSUB"ER?"
1120 '          CMD=&HE0:GOSUB"READ TRACK"    :GOSUB"ER?"
1130 '          CMD=&HF0:GOSUB"WRITE TRACK"   :GOSUB"ER?"
1140 '          CMD=&HD0:GOSUB"FORCE INT"     :GOSUB"ER?"
1150          CMD=-1 :GOSUB"SET LD"          :GOSUB"ER?"
1160          CMD=&H2 :GOSUB"MOTOR"          :GOSUB"ER?"
1170 END
1180 '
1190 LABEL"SET HD"
1200 DUMMY=INP(&HFFE) : '2HD MODE
1210 DUMMY=INP(&HFFD) : 'MFM MODE
1220 STAT=0
1230 RETURN
1240 '
1250 LABEL"SET LD"
1260 DUMMY=INP(&HFFF) : '2D/2DD MODE
1270 DUMMY=INP(&HFFD) : 'MFM MODE
1280 STAT=0
1290 RETURN
1300 '
1310 LABEL"SET FM"
1320 DUMMY=INP(&HFFE) : '2HD MODE
```



```

1330 DUMMY=INP(&HFFC) : 'FM MODE
1340 STAT=0
1350 RETURN
1360 '
1370 LABEL "MOTOR"
1380 OUT &HFFC,CMD
1390 IF CMD AND &H80 THEN GOSUB "WNBSY"
1400 RETURN
1410 '
1420 LABEL "RESTORE"
1430 OUT &HFF8,CMD
1440 GOSUB "WNBSY"
1450 RETURN
1460 '
1470 LABEL "SEEK"
1480 OUT &HFFB,TR : 'DATA REG. (目的のシリンダ番号)
1490 OUT &HFF9,OLDTR : 'TRACK REG. (現在のシリンダ番号)
1500 OUT &HFF8,CMD
1510 GOSUB "WNBSY"
1520 RETURN
1530 '
1540 LABEL "STEP"
1550 OUT &HFF8,CMD
1560 GOSUB "WNBSY"
1570 RETURN
1580 '
1590 LABEL "STEP IN"
1600 OUT &HFF8,CMD
1610 GOSUB "WNBSY"
1620 RETURN
1630 '
1640 LABEL "STEP OUT"
1650 OUT &HFF8,CMD
1660 GOSUB "WNBSY"
1670 RETURN
1680 '
1690 LABEL "READ DATA"
1700 DMA$=HEXCHR$("83 7D FB 0F FF 00 2C 10 80 8D 00 C0 92 CF 87")
1710 GOSUB "SETDMA"
1720 OUT &HFFA,SCT : 'SECTOR REG.
1730 OUT &HFF8,CMD
1740 GOSUB "WNBSY"
1750 GOSUB "RESETDMA"
1760 RETURN
1770 '
1780 LABEL "WRITE DATA"
1790 DMA$=HEXCHR$("83 79 00 C0 FF 00 14 28 80 8D FB 0F 92 CF 05 CF 87")
1800 GOSUB "SETDMA"
1810 OUT &HFF8,CMD
1820 GOSUB "WNBSY"
1830 GOSUB "RESETDMA"
1840 RETURN
1850 '
1860 LABEL "READ ID"
1870 DMA$=HEXCHR$("83 7D FB 0F 05 00 2C 10 80 8D 00 C0 92 CF 87")
1880 GOSUB "SETDMA"
1890 OUT &HFF8,CMD
1900 GOSUB "WNBSY"
1910 GOSUB "RESETDMA"
1920 RETURN
1930 '
1940 LABEL "READ TRACK"
1950 DMA$=HEXCHR$("83 7D FB 0F AF 28 2C 10 80 8D 00 C0 92 CF 87")
1960 GOSUB "SETDMA"
1970 OUT &HFF8,CMD
1980 GOSUB "WNBSY"
1990 GOSUB "RESETDMA"
2000 RETURN
2010 '
2020 LABEL "WRITE TRACK"
2030 DMA$=HEXCHR$("83 79 00 C0 AF 28 14 28 80 8D FB 0F 92 CF 05 CF 87")
2040 GOSUB "SETDMA"
2050 OUT &HFF8,CMD
2060 GOSUB "WNBSY"
2070 GOSUB "RESETDMA"
2080 RETURN
2090 '
2100 LABEL "FORCE INT"
2110 OUT &HFF8,CMD
2120 STAT=INP(&HFF8)
2130 RETURN
2140 '

```



```

2150 LABEL "RESETDMA"
2160   DMA$=HEXCHR$("83"):GOSUB "SETDMA"
2170 RETURN
2180 '
2190 LABEL "WNBSY"
2200   CT=0
2210   IF CT>1000 THEN OUT &HFFC,&H3 AND DN:PRINT "DISK?":STOP
2220   STAT=INP(&HFF8):IF STAT AND &H81 THEN CT=CT+1:GOTO 2210
2230 RETURN
2240 '
2250 LABEL "SETDMA"
2260   FOR I=1 TO LEN(DMA$):OUT &H1F80,ASC(MID$(DMA$,I,1)):NEXT
2270 RETURN
2280 '
2290 LABEL "ER?"
2300   PRINT HEX$(CMD),HEX$(STAT)
2310 RETURN

```

## 2HDの物理フォーマットである

DMAを使ったディスクフォーマットプログラムであるが、リスト9-9を図9-11に従って削除/訂正/追加していただきたい(1010~1170行の削除を忘れずに)。できあがったものがリスト9-8に相当するディスクフォーマットプログラムである。これはturbo BASICでも動くようになっている(もちろんCZ-8FB01でも動く)。RUNするとドライブ番号1(2HD)に入っているディスクに物理フォーマットをかけるようにしてある。打ち込みミスがあると悲惨なことになるので、壊れては困るファイルが入っているディスクはすべてのドライブから抜いておくこと。

さらには2000行以降をリスト9-15と差し替えると、1セクタの容量が512バイトでセ

図9-11 「リスト9-9」→DMAを使った2HDディスクフォーマットプログラムへの変更点

```

1010~1170行を削除

1200 AD=&HC000           : 'フォーマット用データのアドレス
1230 DUMMY=INP(&HFFE)     : '2HDモードにする
1290 FOR CYLINDER=0 TO 76 : 'シリンダ数変更
1440 DUMMY=INP(&HFFE)     : '2HDモードにする
1650 'D2$=USR0(CHR$(9,&HF0,1)):STAT=ASC(D2$):RETURN : '削除する

2000 LABEL "GAP1"
2010 DATA 80,&H4E,12,&H00,3,&HF6,1,&HFC,50,&H4E,0
2020 LABEL "SECTOR"
2030 DATA 12,0,3,&HF5,1,&HFE,4,!
2040 DATA 1,&HF7,22,&H4E,12,0,3,&HF5,1,&HFB,256,&HE5,1,&HF7,54,&H4E,0
2050 LABEL "GAP4"         : '
2060 DATA 598,&H4E,0     : ' <-!!!          !!!          !!!
2070 LABEL "TOTAL SECTOR"
2080 DATA 26             : ' <-!!!
2090 LABEL "SQUE"
2100 DATA 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16
2110 DATA 17,18,19,20,21,22,23,24,25,26
2120 LABEL "TYPE"
2130 DATA 1              : ' <-!!! 1 SECTOR 256 Byte っうことでんがな

```



クタの数が15個、リスト9-16と差し替えると1セクタの容量が1024バイトでセクタの数が8個のフォーマットになる。挙げ句の果てに、リスト9-17はFM(単密度)用のフォーマットである。1235行と1445行もいっしょに付け加えていただきたい。

もちろん少しの訂正で2D、2DDにフォーマットをかけるように変更することも可能である。その点については自由研究である。

#### リスト9-15 2HD 1セクタ512バイトへの変更点

```
2000 LABEL"GAP1"
2010 DATA 80,&H4E,12,&H00,3,&HF6,1,&HFC,50,&H4E,0
2020 LABEL"SECTOR"
2030 DATA 12,0,3,&HF5,1,&HFE,4,!
2040 DATA 1,&HF7,22,&H4E,12,0,3,&HF5,1,&HFB,512,&HE5,1,&HF7,84,&H4E,0
2050 LABEL"GAP4" :'^
2060 DATA 400,&H4E,0 : '<-!!! :!!! :!!!
2070 LABEL"TOTAL SECTOR"
2080 DATA 15 : '<-!!!
2090 LABEL"SQUE"
2100 DATA 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
2110 LABEL"TYPE"
2120 DATA 2 : '<-!!! 1 SECTOR 512 Byte つうことでんがな
```

#### リスト9-16 2HD 1セクタ1024バイトへの変更点

```
2000 LABEL"GAP1"
2010 DATA 80,&H4E,12,&H00,3,&HF6,1,&HFC,50,&H4E,0
2020 LABEL"SECTOR"
2030 DATA 12,0,3,&HF5,1,&HFE,4,!
2040 DATA 1,&HF7,22,&H4E,12,0,3,&HF5,1,&HFB,1024,&HE5,1,&HF7,116,&H4E,0
2050 LABEL"GAP4" :'^
2060 DATA 654,&H4E,0 : '<-!!! :!!! :!!!
2070 LABEL"TOTAL SECTOR"
2080 DATA 8 : '<-!!!
2090 LABEL"SQUE"
2100 DATA 1,2,3,4,5,6,7,8
2110 LABEL"TYPE"
2120 DATA 3 : '<-!!! 1 SECTOR 1024 Byte つうことでんがな
```

#### リスト9-17 2HD 1セクタ128バイトへの変更点

```
1235 DUMMY=INP(&HFFC) : 'FMモードにする
1445 DUMMY=INP(&HFFD) : 'MFMモードにする

2000 LABEL"GAP1"
2010 DATA 40,&HFE,6,&H00,1,&HFC,26,&HFF,0
2020 LABEL"SECTOR"
2030 DATA 6,&H00,1,&HFE,4,!
2040 DATA 1,&HF7,11,&HFF,6,&H00,1,&HFB,128,&HE5,1,&HF7,27,&HFF,0
2050 LABEL"GAP4" :'^
2060 DATA 247,&HFF,0 : '<-!!! :!!! :!!!
2070 LABEL"TOTAL SECTOR"
2080 DATA 26 : '<-!!!
2090 LABEL"SQUE"
2100 DATA 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16
2110 DATA 17,18,19,20,21,22,23,24,25,26
2120 LABEL"TYPE"
2130 DATA 0 : '<-!!! 1 SECTOR 128 Byte つうことでんがな
```



以上のフォーマット用のデータの内訳は、例によって参考文献 10 のお世話になっている。

## グラフィックするのである

DMA の性質としてメモリと I/O は同等に扱えるということである。そこで私の傾向と性格により、リスト 9-14 にリスト 9-18 を追加変更するとたちまちにして FD → G-RAM プログラムができてしまうのである。変更点は、ディスクからリードする場合は転送先を I/O にしたことと、アドレスを G-RAM の始まりの 4000<sub>H</sub> にしたことである。ライトする場合はよって知るべしである。——と書くともっともらしく聞こえるが、実はそれだけでは動かない。というのは、BASIC が G-RAM のバンク 0, 1 を勝手に切り換えるからなのだ。実に残念なことよ。

リスト 9-18 G-RAM アクセス用変更点

1700	DMA\$=HEXCHR\$("83 7D FB 0F FF 00 2C 18 80 8D")
1705	DMA\$=DMA\$+MKI\$(&H4000)+HEXCHR\$("92 CF 87")
1790	DMA\$=HEXCHR\$("83 79")+MKI\$(&H4000)
1795	DMA\$=DMA\$+HEXCHR\$("FF 00 1C 28 80 8D FB 0F 92 CF 05 CF 87")

## ソフト的なフォーマットである

2HD, 2DD のディスクでは容量が違うのであるから、BASIC によるファイル管理も少々違っている。詳しくは turbo の USER'S MANUAL の 187 ページあたりに書いてあるが、大事な点は 2HD でも 1 クラスタは 4K バイトのままで、これは 1 トラック (片面) の容量ではないということである。2D のなごりを引きずっているわけだ。さらにはクラスタ番号である。2DD, 2HD ではクラスタ数がそれぞれ 128 以上あるが、FAT 内では 80<sub>H</sub> ~ 8F<sub>H</sub> はチェーンの終わりを示すものであったから、この番号のクラスタはあってはいけないことになっている。そこで BASIC でどのようにごまかしているかというと、

- 1) 実はクラスタ番号は 2 バイトの値 (16 進数で 4 桁) を持つ
- 2) そして、下 2 桁が 80<sub>H</sub> ~ FF<sub>H</sub> の部分は番号を飛ばす

となっているのである。要するに、2DD, 2HD (8 インチやハードディスクも) では FAT の位置 (レコード番号) やディレクトリの位置が違うということである。

## さらにグラフィックするのである

リスト 9-19 である。このプログラムはリスト 9-14 + リスト 9-16 で作った、1 セクタが 1024 バイトのフォーマットのディスクを画像用にしてしまうものである。具体的に言うと、1 トラック (片面) に 8 セクタあり、1 シリンダ (裏表で計 2 トラック) が 16K バイ



トであることから、3 シリンダに 48K バイト分のグラフィックデータを書き込んでしまうのである。ディスクアクセスにはマルチセクタを使っている。FDC の説明の最初の所でも書いたが、マルチセクタを使うと FDC は「RECORD NOT FOUND」でエラーを起こすまで BUSY のままである。そこでここでは DMA の「エンド・オブ・ブロックで割り込み」の機能を使っている。これにより FDC が第 9 セクタ（ここでは存在しない）を捜してエラーを起こすのを待たずに済むようになっている。最初にある MEM\$ は、FDC のステータスレジスタを読み出すルーチン+割り込み処理ルーチンである。「FDC のステータスをチェックするなら BASIC の INP 関数を使えばよい」と思うだろうが、実はそこには恐ろしいワナが隠されている。なんと、私の調べたところによると、INP 関数を使うと BASIC は、

### 勝手に G-RAM の入力ページ（バンク 0, 1）をひっくり返してしまう

のだ。よって CALL 命令で呼び出した機械語プログラムによって FDC のステータスをチェックしなければならないのだ。もっともこの部分は徹底的に探ったわけではないから、いまいち確信はないのだが。ま、とにかくそのようなわけなのである。

#### リスト 9-19 48K G-RAM ロード/セーブプログラム

```

100 CLEAR &HEF00
110 MEM$(&HEF00,16)=HEXCHR$("01 F8 0F ED 78 32 26 EF C9 F3 F5 C5 01 80 1F 3E")
120 MEM$(&HEF10,16)=HEXCHR$("AF ED 79 3E 8B ED 79 3E AB ED 79 3E 01 32 25 EF")
130 MEM$(&HEF20,7)=HEXCHR$("C1 F1 FB ED 4D 00 00")
140 INIT
150 DN=1
160 DUMMY=INP(&HFFE) : '2HD(1.6) MODE
170 MEM$(&HF814,2)=MKI$(&HEF09) : 'OLD VALUE=&H81C3
180 '
190 GOSUB"RESTORE":OLDTR=0
200 FOR TTR=0 TO 72 STEP 3
210   CLS4
220   SYMBOL(32,0),STR$(TTR),20,10,7,0,PSET
230   GOSUB"WRITE"
240 NEXT
250 '
260 GOSUB"RESTORE":OLDTR=0
270 FOR TTR=0 TO 72 STEP 3
280   CLS4
290   GOSUB"READ"
300 NEXT
310 '
320 OUT &HFFC,&H3 AND DN
330 DUMMY=INP(&HFFF) : '2D,2DD(0.5M,1M) MODE
340 MEM$(&HF814,2)=MKI$(&H81C3)
350 END
360 '
370 LABEL"READ"
380 CCC=&H90
390 DD1$=HEXCHR$("83 7D FB 0F FF 1F 2C 18 9D")
400 DD2$=HEXCHR$("32 10 92 A0 CF 8B 87")
410 GOSUB"EXEC"
420 RETURN
430 '
440 LABEL"WRITE"
450 CCC=&HB0
460 DD1$=HEXCHR$("83 79")
470 DD2$=HEXCHR$("FF 1F 1C 28 9D FB 0F 32 10 92 CF 05 A0 CF 87")
480 GOSUB"EXEC"
490 RETURN
500 '
510 LABEL"EXEC"
520 GAD=&H4000
530 FOR TR=TTR TO TTR+2
540   GOSUB"SEEK":OLDTR=TR

```



```

550   FOR SIDE=0 TO 1
560     DMA$=DD1$+MKI$(GAD)+DD2$
570     GOSUB"SETDMA"
580     OUT &HFFC,&H80 OR DN OR &H10*SIDE:GOSUB "WNBSY" : 'MOTOR ON
590     OUT &HFFA,1 : 'SECTOR #
600     POKE &HEF25,0 : 'CLEAR FLAG
610     OUT &HFF8,CCC : 'COMMAND
620     CALL &HEF00
630     IF (PEEK(&HEF26) AND 1)=0 THEN 660
640     IF PEEK(&HEF25) THEN 660
650     GOTO 620
660     PRINT PEEK(&HEF25),PEEK(&HEF26),INP(&HFF8)
670     OUT &HFF8,&HD0
680     GAD=GAD+&H2000
690   NEXT
700 NEXT
710 DMA$=HEXCHR$("C3 C3 C3 C3 C3 C3 83 80"):GOSUB"SETDMA"
720 RETURN
730 '
740 LABEL"WNBSY"
750 CT=0
760 IF CT>1000 THEN OUT &HFFC,&H2:PRINT"DISK?":STOP
770 IF INP(&HFF8) AND &H81 THEN CT=CT+1:GOTO 760
780 RETURN
790 '
800 LABEL"SETDMA"
810 FOR I=1 TO LEN(DMA$):OUT &H1F80,ASC(MID$(DMA$,I,1)):NEXT
820 RETURN
830 '
840 LABEL"SEEK"
850 OUT &HFFB,TR : 'データレジスタ
860 OUT &HFF9,OLDTR : 'トラックレジスタ
870 OUT &HFF8,&H10 : 'シークコマンド
880 GOSUB"WNBSY"
890 RETURN
900 '
910 LABEL"RESTORE"
920 OUT &HFFC,&H80 OR DN:GOSUB "WNBSY": 'MOTOR ON
930 OUT &HFF8,&H0 :GOSUB "WNBSY": 'HOME
940 RETURN

```

それでリスト 9-19 でやっていることだが、中心は 370 行以降の "READ" と "WRITE" である。それぞれのルーチンは変数 TTR (Top Track) から始まる 3 シリンダと 48K バイトの G-RAM の間でデータの読み書きを行なう。プログラム全体では、最初は SYMBOL 文を使って G-RAM に TTR の値をでかでかと書き、それを次々とディスクに書き込み、次には、次々に読み出して表示している。これは実行させてみれば一目瞭然であろう。そこで 110~130 行の MEM\$~による機械語プログラムを簡単に説明しておく。アセンブルリストは、リスト 9-20 である。EF00<sub>H</sub>~EF08<sub>H</sub>は単に FDC のステータスを読み出して EF26<sub>H</sub>番地に格納しているだけである。大事なのは EF09<sub>H</sub>~で、これは「エンド・オブ・ブロック」後の DMA の再初期化と「フラグ立て」(EF25<sub>H</sub>番地)を行なっている。このフラグが立ったなら、DMA が 8K バイトの転送を終えたわけだから、FDC の空しい第 9 セクタ探しをフォースインタラプトで打ち切ってやるわけである。

リスト 9-20 「リスト 9-19」の機械語部分

			.280	
			.PHASE	0EF00H
		;		
0FF8		FDC	EQU	0FF8H ;FDC CMD AND STAT
1F80		DMA	EQU	1F80H ;DMA I/O ADDR.
		;		
EF00	01 0FF8	START:	LD	BC,FDC
EF03	ED 78		IN	A,(C)
EF05	32 EF26		LD	(STAT),A



EF08	C9		RET
		;	
EF09	F3	DMAINT:	DI
EF0A	F5		PUSH AF
EF0B	C5		PUSH BC
EF0C	01 1F80		LD BC,DMA
EF0F	3E AF		LD A,0AFH ;WR6:DISABLE INT.
EF11	ED 79		OUT (C),A
EF13	3E 8B		LD A,08BH ;WR6:REINIT. STAT
EF15	ED 79		OUT (C),A
EF17	3E AB		LD A,0ABH ;WR6:ENABLE INT.
EF19	ED 79		OUT (C),A
EF1B	3E 01		LD A,1
EF1D	32 EF25		LD (INTF),A;SET FLAG
EF20	C1		POP BC
EF21	F1		POP AF
EF22	FB		EI
EF23	ED 4D		RETI
		;	
EF25		INTF:	DS 1
EF26		STAT:	DS 1
		;	
			END

以上が DMA でディスクを使ってみた例である。残念ながら 8 インチやハードディスクなどは手付かずとなってしまったが、さっさと次の章に行ってしまうのであった。



第

10

章

PSG



PSGは基本である



## ■ 第10章

# PSGは基本である・・・・・・・・・・

この章では PSG をやるのである。PSG というのは、Programmable Sound Generator, すなわち「プログラム可能な音声発生機」なのである。プログラム可能とはいっても別にたいしたことはないからそのつもりで。

X1 で使われている PSG は AY-3-8910 という、もう古典と言ってもよい LSI である。ただし turboZ などでは、それとコンパチな YAMAHA の YM2149 も使われている(ちなみに YM2149 は「SSG」と呼ばれている。このことから逆に AY-3-8910 を SSG と呼んだりもする)。

で、この石を簡単に紹介すると、

- 1) 音量調節付きで 3 重和音が使える。
- 2) ノイズ(雑音)が使える。
- 3) エンベロープと呼ばれる機能により、指定した周期、形状(8 パターン)で音量を変化させることができる。
- 4) 2 ポートの 8 ビットパラレルポートが使える。X1 ではこれをジョイスティックに使用している。

などとなる。

AY-3-8910 から 4) の機能を取り去ってピン数を減らしたものは型番が AY-3-8913 と呼ばれ、MZ-5500/6500 で使われている。FM-7 では 8910, 8913 が混在しているそうである。どっちにしても、FM-7 ではパラレルポートを使ってジョイスティックをサポートしていないから同じことである。

AY-3-8910 の使い方は簡単で、基本的には 16 個あるレジスタにそれなりのデータを書き込んでやったり、もしくは読み出したりすればよい。この点については第 1 章の CRTC と似ている。そこで早速 PSG のレジスタ表が表 10-1 である。

最初に言うておくが、 $R_0$  と  $R_1$ ,  $R_2$  と  $R_3$ ,  $R_4$  と  $R_5$  は別々に見ずに「12 ビットの値を下 8 ビットと上 4 ビットに分けた」と見るべきである。これは  $R_{11}$  と  $R_{12}$  でも同じで、こちらの方は 16 ビットの値を上下 8 ビットずつに分けたものとみなすことができる。この点を注意しつつ、順に解説する。

### ① $R_0$ , $R_1$ (チャンネル A トーン周波数)

$R_1$  を上 4 ビット,  $R_0$  を下 8 ビットとして作る 12 ビットの整数を  $A_f$  とする。 $A_f$  は 0 ~ 4095 である。すなわち、

$$A_f = R_0 + R_1 \times 256 \cdots \text{式 1}$$

この  $A_f$  とチャンネル A から出る音の周波数の関係は、音の周波数を  $f_a$  (Hz) とすると、

$$f_a = \frac{2 \times 10^6}{16 \times A_f} = \frac{125000}{A_f} \cdots \text{式 2}$$



表 10-1 PSG レジスタ表

レジスタ番号	レジスタ機能	ビット構成								データ
		D7	D6	D5	D4	D3	D2	D1	D0	
R0	チャンネル A 周波数	下位 8 ビット								0 ~ 255
R1						上位 4 ビット				0 ~ 15
R2	チャンネル B 周波数	下位 8 ビット								0 ~ 255
R3						上位 4 ビット				0 ~ 15
R4	チャンネル C 周波数	下位 8 ビット								0 ~ 255
R5						上位 4 ビット				0 ~ 15
R6	ノイズ周波数				5 ビット					0 ~ 31
R7	チャンネル選択	IN/OUT		ノイズ			トーン			0~255 (D0~D5 は、0 を設定すると選択される)
		IOB	IOA	C	B	A	C	B	A	
R8	チャンネル A 音量				M	4 ビット				0 ~ 16
R9	チャンネル B 音量				M	4 ビット				0 ~ 16
R10	チャンネル C 音量				M	4 ビット				0 ~ 16
R11	エンベロープ周期	下位 8 ビット								0 ~ 255
R12		上位 8 ビット								0 ~ 255
R13	エンベロープ形状					4 ビット				0 ~ 15
R14	JOY 1 (IOA)	8 ビット								0 ~ 255
R15	JOY 2 (IOB)	8 ビット								0 ~ 255

となっている。**式 2**で「 $2 \times 10^6$ 」と「16」という数値がいきなり出てくるが、 $2 \times 10^6$ というのは、LSI に供給されているクロックの周波数 (2MHz) である。だからハードウェアが異なれば、この部分の値は変わってくる。M (メガ) というのは  $10^6 = 1000000$  の単位なのだ、念のため。次に出てくる 16 は、AY-3-8910 を設計した人が勝手に決めた数値である (と思う)。なお、**式 2**より  $A_f$  の値が大きいほど周波数の小さい音 (低い音) が出ることが分かる。要するに逆比例なのだ。

**式 1, 2**より「周波数  $f$  (Hz) の音を出したいときに、どういう数値を  $R_0$ ,  $R_1$  に書き込めばよいか」を計算できる。

まず、**式 2**から、

$$A_f = \frac{125000}{f_a}$$

が分かる。ただし  $A_f$  は整数じゃないとまずいので、小数点第 1 位で四捨五入するのがよいだろう。そこで、本当は、

$$A_f = \text{INT}\left(\frac{125000}{f_a} + 0.5\right)$$

となる。次に  $A_f$  から  $R_0$ ,  $R_1$  を計算するわけだが、これは簡単に、

$$R_0 = A_f \bmod 256$$

$$R_1 = \text{INT}\left(\frac{A_f}{256}\right)$$



となる。めでたしめでたし。ただし今のうちに言っておくが、 $R_0$ 、 $R_1$ だけをセットしても音は出てこない。注意するように。なお、やってみると分かるが、 $A_1$ を0にした場合、式の上では周波数が無限大の音が出ることになる。自然界では当然そんなことはあり得ないので、 $A_1=0$ の場合は考える必要はない。また、音質を良くするためにも電気回路では、あまり高い周波数（成分）の音は増幅されないのが普通だから、そもそもスピーカーから出ていないはずである。

さてさて、音の高さについて書いてきたのであるが、周波数ばかりで説明していても始まらない。実用的にはドレミの音階が必要である。そこで表 10-2 が、えーとえーと、「1939 年 5 月 ロンドンにおける国際会議で規定された音階表」である。これは『理科年表』に載っていたものである。

表 10-2 国際基準イ =  $a^1$  = 440Hz に基づく十二平均律音階 (単位は Hz)

	$C_2$	$C_1$	C	c	$c^1$	$c^2$	$c^3$	$c^4$	$c^5$
C	16.352	32.703	65.406	130.81	261.63	523.25	1046.5	2093.0	4186.0
C #	17.324	34.648	69.296	138.59	277.18	554.37	1108.7	2217.5	4434.9
D	18.354	36.708	73.416	146.83	293.66	587.33	1174.7	2349.3	4698.6
D #	19.445	38.891	77.782	155.56	311.13	622.25	1244.5	2489.0	4978.0
E	20.602	41.203	82.407	164.81	329.63	659.26	1318.5	2637.0	5274.0
F	21.827	43.654	87.307	174.61	349.23	698.46	1396.9	2793.8	5587.7
F #	23.125	46.249	92.499	185.00	369.99	739.99	1480.0	2960.0	5919.9
G	24.500	48.999	97.999	196.00	392.00	783.99	1568.0	3136.0	6271.9
G #	25.957	51.913	103.83	207.65	415.30	830.61	1661.2	3322.4	6644.9
A	27.500	55.000	110.00	220.00	440.00	880.00	1760.0	3520.0	7040.0
A #	29.135	58.270	116.54	233.08	466.16	932.33	1864.7	3729.3	7458.6
B	30.868	61.735	123.47	246.94	493.88	987.77	1975.5	3951.1	7902.1

(1979年度版理科年表より)

この表には厳格な規則性がある。中央に四角で囲んだ数字があるが、これが  $a^1 = \text{イ}$  と呼ばれる音で、何がなんでもここは 440Hz なのである。これはラの音である。X1 で出すのなら、

PLAY "O4A"

で出てくる。この音を基準として、半音上げた音 (#A) の周波数は、440 に  $^{12}\sqrt{2} = 2^{\frac{1}{12}} = 1.05946 \dots$  (2 の 12 乗根) を掛けたもの、逆に半音下げた音 (#G) は 440 を  $^{12}\sqrt{2}$  で割ったものである。同じように  $^{12}\sqrt{2}$  で掛けたり割ったりすると、そのたびに半音上がったり下がったりする。1 オクターブは半音が 12 個分だから、1 オクターブ上の音は、ちょうど周波数が 2 倍になる ( $^{12}\sqrt{2}$  の 12 乗は 2 なのだ)。逆に 1 オクターブ下の音は、周波数がちょうど 2 分の 1 である。表 10-2 を見て納得していただきたい。なお、なぜこのように決められているのか不思議に思う人もいるだろうが、それは人間の聴覚システムがそうなっているからなのである。それ以上のことは私の知ったことではない。

では、AY-3-8910 で音階表どおりの周波数の音を出すにはどうすればよいかという問題に移る。真面目に計算してもよいのだが、私は turbo BASIC の MUSIC@ 命令を使ったのである。MUSIC 命令だと、音が出ている間はずっと待たされてしまうが、MUSIC@ 命令



ならば待たされることなく次の命令を実行してくれる。だからそのときに PSG の  $R_0$  と  $R_1$  を読み出せばよい。なお、それじゃ MUSIC@ 命令はいつ音を止めるのかというと、タイマ割り込みを使っているのである。だから心配せずとも、時期が来れば止まってくれる。**表 10-3** が  $R_0$  と  $R_1$  の値を表にしたものである。16 進数で、「 $R_0 + R_1$ 」の形式でできている。四角で囲んであるところが 440 Hz に対応する所である。とはいっても、本当に正確に 440 Hz を出せるわけではない。なにせ  $R_0, R_1$  は整数に限られるのである。これはほかの音階でも言えることで、特に高い音になるほどずれは大きくなる。まあ、こういうものだと思うしかないのである。なお、1 オクターブ上げる = 周波数を 2 倍する =  $A_1$  を半分にする、であることに注意。

表 10-3 各音階に対するレジスタの値（「 $R_0$ 」+「 $R_1$ 」の形式）

	C	#C	D	#D	E	F	#F	G	#G	A	#A	B
01	EF+0E	17+0E	4D+0D	8E+0C	DA+0B	2F+0B	8F+0A	F7+09	68+09	E1+08	62+08	E9+07
02	77+07	0B+07	A6+06	47+06	ED+05	97+05	47+05	FB+04	B4+04	70+04	31+04	F4+03
03	BB+03	85+03	53+03	23+03	F6+02	CB+02	A3+02	7D+02	5A+02	38+02	18+02	FA+01
04	DD+01	C2+01	A9+01	91+01	7B+01	65+01	51+01	3E+01	2D+01	<u>1C+01</u>	0C+01	FD+00
05	EE+00	E1+00	D4+00	C8+00	BD+00	B2+00	A8+00	9F+00	96+00	8E+00	86+00	7E+00
06	77+00	70+00	6A+00	64+00	5E+00	59+00	54+00	4F+00	4B+00	47+00	43+00	3F+00
07	3B+00	38+00	35+00	32+00	2F+00	2C+00	2A+00	27+00	25+00	23+00	21+00	1F+00
08	1D+00	1C+00	1A+00	19+00	17+00	16+00	15+00	13+00	12+00	11+00	10+00	0F+00

②  $R_2, R_3$  (チャンネル B トーン周波数)

③  $R_4, R_5$  (チャンネル C トーン周波数)

②, ③はチャンネル A が B, C になっただけで、それ以外はまったく同じである。

④  $R_6$  (ノイズ周波数)

AY-3-8910 には前述したようにノイズを出す機能がある。といっても別にノイズ専門のチャンネルがあるわけではなく、チャンネル A~C のうちから割り当てるのである。割り当て方については⑤で説明する。

ノイズとはどんなときに使うのかといえば、波の音とか爆発音などの効果音ということになるだろう。 $R_6$  はそのノイズの周波数を決めるものである。だが、ノイズというのは、一定の幅の周波数帯の音を含んでいる音のことだから、大体の範囲を決めるということである。具体的には、周波数が高ければ「シャー」、低ければ「ザー」という感じの音になる。 $R_6$  とノイズの高さは、例によって逆比例の関係になる。つまり、 $R_6$  の値が小さいほどノイズの音は高くなる。

⑤  $R_7$  (ミキサー, I/O コントロール)

この部分はフラグの集まりと考えるのが手っ取り早い。まず、 $D_0 \sim D_5$  はチャンネル A, B, C から何を出すのかの指定である。チャンネル A を指定するのは  $D_3$  と  $D_0$  である。**図 10-1** に  $D_3$  と  $D_0$  の値 (1/0) とチャンネル A からの出力の関係を示す。

**図 10-1** はほかのチャンネルにとっても同じことが言える。1 で OFF, 0 で ON と変態的に負論理になっていることに注意。

$D_6, D_7$  は二つある I/O ポートを入力に使うか出力に使うかの指定である。もちろん二つとも別々に指定することができる。0 で入力に指定, 1 で出力に指定である。X1 では通常は二つともジョイスティックからの入力になっているから、ともに 0 である。なお、実際



図 10-1 ノイズ/トーン指定ビット  
(チャンネル A の場合)

D <sub>3</sub>	D <sub>0</sub>	チャンネル A 出力
0	0	ノイズ+トーン
0	1	ノイズ
1	0	トーン
1	1	なし

のジョイスティックポートのデータの受け渡しは R<sub>14</sub>, R<sub>15</sub>を使う。

⑥ R<sub>8</sub>, R<sub>9</sub>, R<sub>10</sub> (チャンネル A～C 音量, エンベロープ指定)

R<sub>8</sub>, R<sub>9</sub>, R<sub>10</sub>のそれぞれ下 4 ビットの D<sub>3</sub>～D<sub>0</sub>は音量を指定する。4 ビットだから 0 (最小) ～15 (最大) までの 16 段階である。ただし D<sub>4</sub>が 1 ならば, そのチャンネルはエンベロープモードになり, D<sub>3</sub>～D<sub>0</sub>の値は無視される。この場合の音量は, R<sub>11</sub>, R<sub>12</sub>, R<sub>13</sub>の指定するエンベロープ周期と, エンベロープパターンに支配される。

⑦ R<sub>11</sub>, R<sub>12</sub> (エンベロープ周期設定)

これは, 次で指定するエンベロープのパターンの周期を設定するものである。エンベロープというのは, 図 10-2 にあるように 8 種のパターンがある。エンベロープ周期は図中の t<sub>e</sub>の長さだと思えばよい。

$$A_e = R_{11} + R_{12} \times 256$$

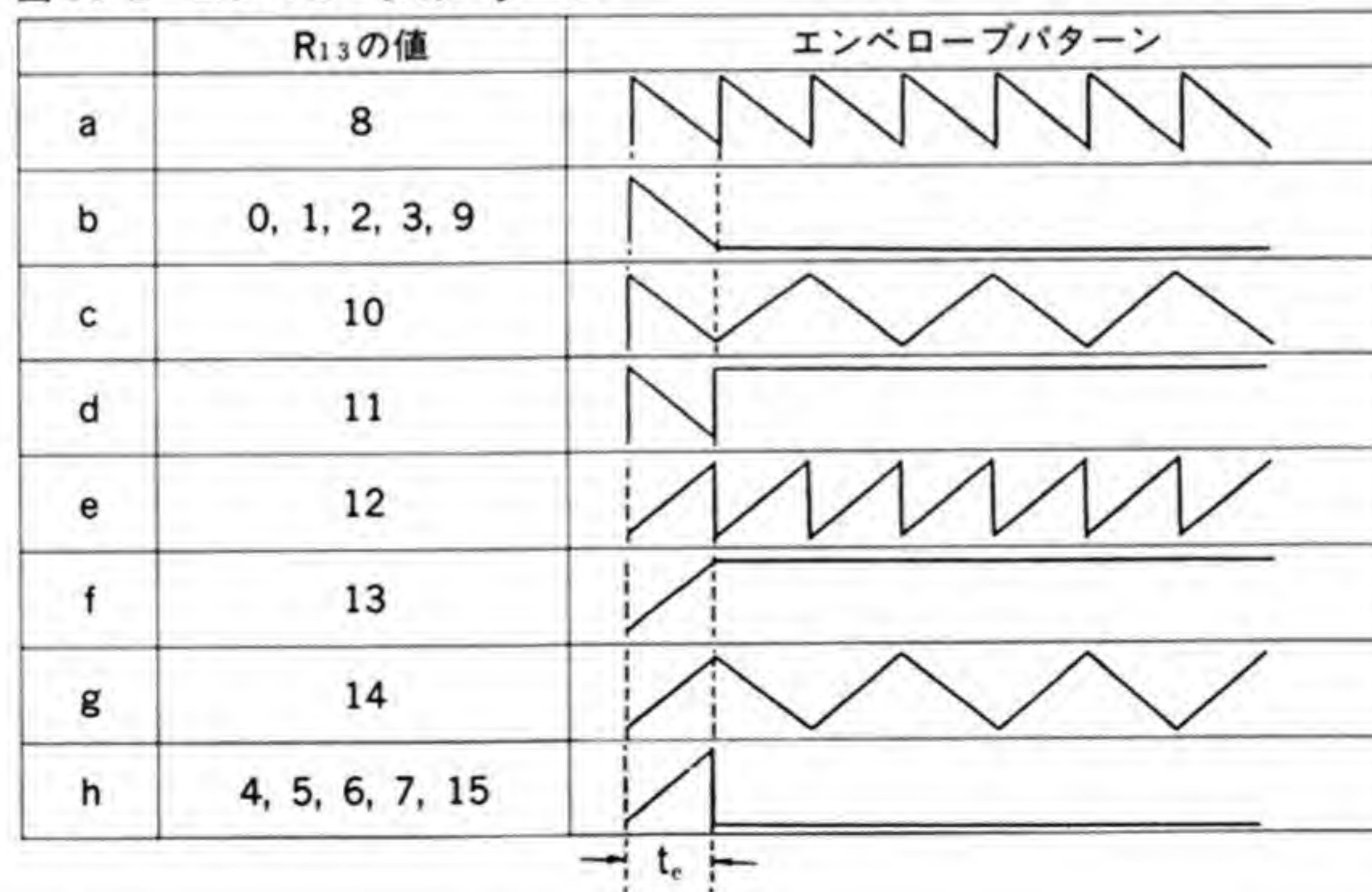
とすると,

$$t_e = \frac{256 \times A_e}{2 \times 10^6}$$

で計算できる。

エンベロープというのは音量を自動的に変えるもので, たとえば「1 秒間だけ音を出して, その後は出さない」などということを, PSG が勝手にやってくれる機能である。図 10-2 のエンベロープパターンを見れば分かるように, 音量アップ/ダウンを延々と繰り返すパターンもある。

図 10-2 エンベロープのパターン





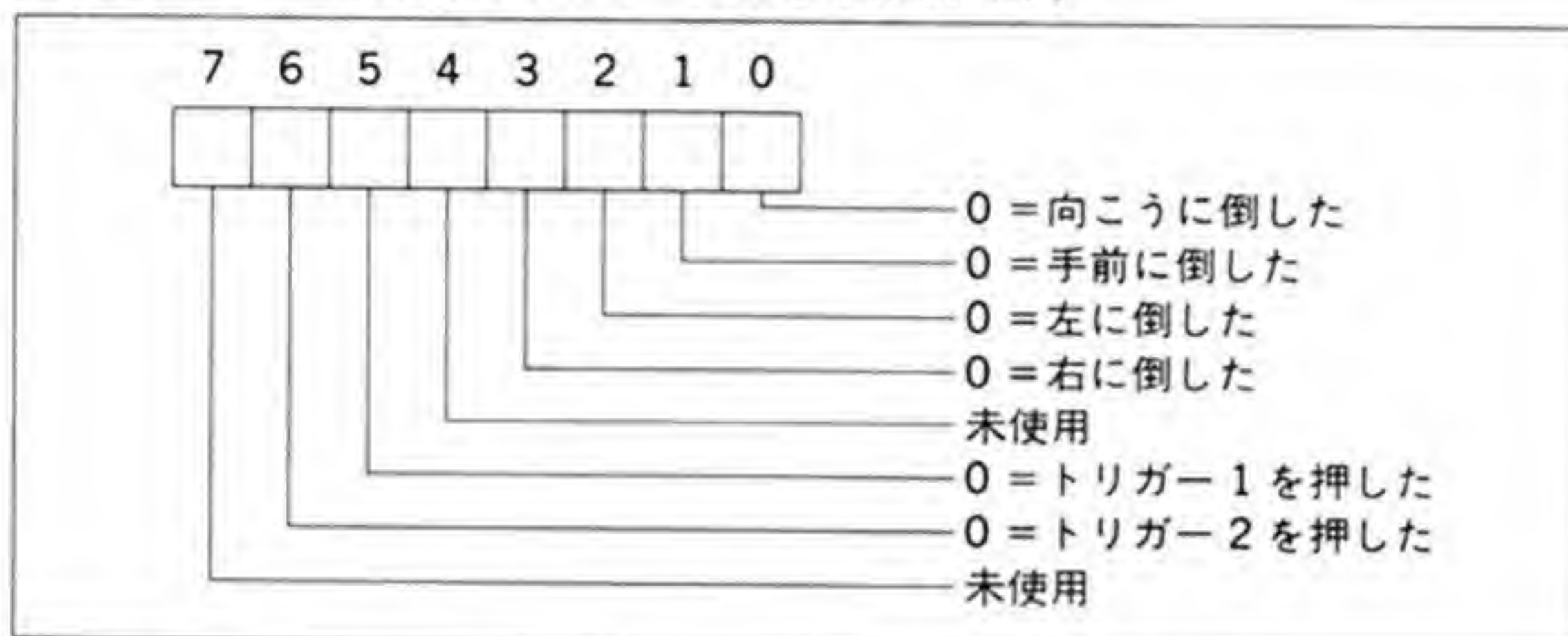
### ⑧ R<sub>13</sub> (エンベロープパターン指定)

図 10-2 に示すエンベロープパターンのうち、どれを使うかを指定するものである。パターン b, h は複数の R<sub>13</sub> の値に対応しているが、「R<sub>13</sub> には 8～15 の値だけを設定するものだ」と思っていればよいのだ。

### ⑨ R<sub>14</sub>, R<sub>15</sub> (I/O ポート入出力)

前にも書いたように、X1 ではジョイスティックポートの入出力に使われている。R<sub>14</sub> がジョイスティック 1, R<sub>15</sub> がジョイスティック 2 である。図 10-3 参照。

図 10-3 ジョイスティックのデータの意味(R<sub>14</sub>, R<sub>15</sub>)



### ⑩ 最後に PSG へのアクセス法である。

まず、レジスタ番号を指定する。つまり、  
OUT &H1C00, レジスタ番号 (0～15)  
の後に、  
OUT &H1B00, データ  
で、データを書き込める。逆にデータを読み出すなら、  
~ = INP (&H1B00)  
となる。

以上で基礎編の説明を終えたわけである。そして、ここでサンプルプログラムが発動するのだ。

## サンプルなのである

### リスト 10-1 : ドミノ

表 10-2 を使えば一発で分かるはずである。チャンネル A, B, C にそれぞれドミノを割り当て、R<sub>7</sub> で三つともトーンのみを指定。後は音量を三つとも 15 にしておしまいである。

#### リスト 10-1 ドミノ

```
100 MUSIC "R0"  
110 SOUND 0,&HDD  
120 SOUND 1,&H1 : 'ト'  
130 SOUND 2,&H7B
```



```

140 SOUND 3,&H1 : 'ミ
150 SOUND 4,&H3E
160 SOUND 5,&H1 : 'ソ
170 SOUND 7,&B111000
180 SOUND 8,15
190 SOUND 9,15
200 SOUND 10,15

```

さて、このプログラムが実にうっとうしいと思うわけである。そこでリスト 10-2 である。まず、サブルーチン“QUIET”は PSG を黙らせるサブルーチン、“XSOUND”は PSG のレジスタに指定されたデータを次々と設定するサブルーチンである。“QUIET”を使うには、ただ呼ぶだけでよい。“XSOUND”は少々複雑である。使うには、まず DATA 文を用意する。約束ごとは次の五つである。

- 1) 最初は、レジスタ番号をデータとする。たとえば「1」ならば R<sub>1</sub> から設定を始めることになる。
- 2) レジスタに設定したいデータは、0～255 まで、&H～、&B～などもよい。
- 3) 設定したくないレジスタに対応するデータは「S」とする。つまり R<sub>4</sub> と R<sub>6</sub> だけを 0 にして、R<sub>5</sub> をそのままにしておきたければ、

```
DATA 4, 0, S, 0, .....
```

とすればよい。

- 4) DATA 文の最後は「EOD」(End Of Data) とする。

- 5) GOSUB “XSOUND” の直前に、

```
RESTORE (データ文の行番号、もしくはラベル)
```

を実行しておく。

てなところである。もっとも、それは X1 ユーザーに限ってのことで、turbo ではサウンド文が拡張されていて、“XSOUND”に相当することがそのまま SOUND 文で実行できるから、きっと、むっとしているはずである。人生とはそんなものである。ところで、リスト 10-2 で行番号がいきなり 1000 に飛んでいることから分かるように、これらのサブルーチンはほかのリストでも流用される運命にある。心得ておいていただきたい。

#### リスト 10-2 PSG 用サブルーチン

```

100 GOSUB "QUIET"
110 RESTORE 120:GOSUB "XSOUND"
120 DATA 0,&HDD,&H01,&H7B,&H01,&H3E,&H01,S,&B111000,15,15,15,EOD
130 PAUSE 10
140 GOSUB "QUIET"
150 END
160 '
1000 LABEL "QUIET"
1010 SOUND 8,0:SOUND 9,0:SOUND 10,0
1020 RETURN
1030 '
1040 LABEL "XSOUND"
1050 READ C
1060 READ A$:IF A$="EOD"THEN RETURN
1070 IF A$="S" THEN GOTO1090
1080 SOUND C,VAL(A$)
1090 C=C+1:GOTO1060

```



### リスト 10-3 : 銃声

エンベロープパターンは b を使っている。エンベロープ周期は  $R_{11} = 0$ ,  $R_{12} = 20$  だから、

$$t_e = \frac{256 \times (20 \times 256 + 0)}{2 \times 10^6} \approx 0.66 (\text{秒})$$

となる。つまり、最初はボリューム最大で、0.66 秒の間に 0 まで減少するのである。3 チャンネルともノイズモードにしてある。130 行を見ても分かるように、何かキーを押すと音が出る。押し続けると、機関銃となるわけである。

#### リスト 10-3 銃声

```
100 GOSUB"QUIET"
110 RESTORE 120:GOSUB"XSOUND"
120 DATA 6,20,&B000111,16,16,16,0,20,9,EOD
130 IF INKEY$(0)="" THEN 130 ELSE 110
140 END
150 '
1000 LABEL"QUIET"
1010 SOUND 8,0:SOUND 9,0:SOUND 10,0
1020 RETURN
1030 '
1040 LABEL"XSOUND"
1050 READ C
1060 READ A$:IF A$="EOD"THEN RETURN
1070 IF A$="S" THEN GOTO1090
1080 SOUND C,VAL(A$)
1090 C=C+1:GOTO1060
```

### リスト 10-4 : UFO

エンベロープパターンは g で、3 チャンネルともトーン。この手の音を作るのは楽なのである。

#### リスト 10-4 UFO

```
100 GOSUB"QUIET"
110 RESTORE 120:GOSUB"XSOUND"
120 DATA 0,100,0,102,0,200,0,S,&B111000,16,16,16,10,5,14,EOD
130 PAUSE 50
140 GOSUB"QUIET"
150 END
160 '
1000 LABEL"QUIET"
1010 SOUND 8,0:SOUND 9,0:SOUND 10,0
1020 RETURN
1030 '
1040 LABEL"XSOUND"
1050 READ C
1060 READ A$:IF A$="EOD"THEN RETURN
1070 IF A$="S" THEN GOTO1090
1080 SOUND C,VAL(A$)
1090 C=C+1:GOTO1060
```

### リスト 10-5 : SL とヘリコプター

RUN すると、最初に SL の音を出す。次にヘリコプターの音である。ヘリコプターの方は、チャンネル B がノイズだけである。そのつもりになって聞くと、そう聞こえるものな



のである。

#### リスト 10-5 SL とヘリコプター

```
100 GOSUB"QUIET"
110 RESTORE 120:GOSUB"XSOUND"
120 DATA 6,10,&B110111,16,0,0,0,6,12,EOD
130 PAUSE 50
140 RESTORE 150:GOSUB"XSOUND"
150 DATA 0,0,5,0,0,0,0,5,&B100110,16,16,0,0,1,10,EOD
160 PAUSE 50
170 GOSUB"QUIET"
180 END
190 '
1000 LABEL"QUIET"
1010 SOUND 8,0:SOUND 9,0:SOUND 10,0
1020 RETURN
1030 '
1040 LABEL"XSOUND"
1050 READ C
1060 READ A$:IF A$="EOD"THEN RETURN
1070 IF A$="S" THEN GOTO1090
1080 SOUND C,VAL(A$)
1090 C=C+1:GOTO1060
```

#### リスト 10-6 : 波の音

これはよく見かけるサンプルであるが、リスト 10-7 のように、ランダムにエンベロープ周期を変えてやると本物に近くなる。時間かせぎの PAUSE 文が 140 行にあるが、別にエンベロープ周期にセットする値と関係あるわけではない。何をしても音がずたずたになるわけではないから、適当でよいのだ。

#### リスト 10-6 波の音

```
100 GOSUB"QUIET"
110 RESTORE 120:GOSUB"XSOUND"
120 DATA 6,25,&B110111,16,0,0,0,99,14,EOD
130 END
140 '
1000 LABEL"QUIET"
1010 SOUND 8,0:SOUND 9,0:SOUND 10,0
1020 RETURN
1030 '
1040 LABEL"XSOUND"
1050 READ C
1060 READ A$:IF A$="EOD"THEN RETURN
1070 IF A$="S" THEN GOTO1090
1080 SOUND C,VAL(A$)
1090 C=C+1:GOTO1060
```

#### リスト 10-7 ランダムな波の音

```
100 GOSUB"QUIET"
110 RESTORE 120:GOSUB"XSOUND"
120 DATA 6,25,&B110111,16,0,0,0,99,14,EOD
130 T=40+INT(RND(1)*60):PRINTT
140 SOUND 12,T:PAUSE T:GOTO130
150 '
1000 LABEL"QUIET"
1010 SOUND 8,0:SOUND 9,0:SOUND 10,0
1020 RETURN
```



```
1030 '  
1040 LABEL "XSOUND"  
1050 READ C  
1060 READ A$: IF A$="EOD" THEN RETURN  
1070 IF A$="S" THEN GOTO 1090  
1080 SOUND C, VAL(A$)  
1090 C=C+1: GOTO 1060
```

以上でサンプルの紹介は終わるが、ここまで見てきて、疑問を持った人もいるはずである。それは何かというならば、エンベロープのパターンはいつ始まるかということである。R<sub>13</sub>(エンベロープパターン)に値をセットしたときからなのか？ それとも、R<sub>8</sub>~R<sub>10</sub>のD<sub>5</sub>を1にセットした瞬間からなのか？ ということである。すなわち、エンベロープを発動するトリガー(引金)は何なのかである。

答えはR<sub>13</sub>である。ここに値を書き込まれた瞬間から、エンベロープパターンに従って音が出る。これに対してR<sub>11</sub>, R<sub>12</sub>のエンベロープ周期は単なるカウンタの周期ぐらいの意味しかない。だからリスト 10-7 のランダムな波の音は、それらしく聞こえたのである。

というところで PSG は終わりである。最近では FM 音源が花盛りであるが、PSG も使い方によっては非常に強力なので、きちんと押さえておいて欲しいのであった。







第

11

章

# FM音源



FM音源ナハトムジーク



## ■ 第11章

# FM音源ナハトムジーク・・・・・・・・

この章ではFM音源についてやってしまうのである。となれば目的はMMLである。で、MMLであるが、turboではNEW turbo BASICにより手軽にFM音源を楽しめるようになった。しかしX1ではそうはいかない。てなわけで、ここでサポートするしだいである。ちなみにOh! MZの連載時にはあちこちにバグが入ったりしたが、それらをすべて直してある(はずである)。

さて、X1のFM音源ボードといえはCZ-8BS1なのであるが、このボードの主演はYM2151、別名OPMと呼ばれるLSIなのである。それに対して他の大概のパソコンに使われているFM音源のLSIはYM2203(OPN)なのである。この両者の違いはというと、

- OPMはFM音源を8チャンネル持っており、それぞれのチャンネルごとに左右(もしくは両方)出力を選択できる。またLFO機能が付いている。
- OPNはFM音源を3チャンネルと、SSG音源を3チャンネル(AY-3-8910と同じ機能)持っている。出力はモノラルのみ。LFO機能は付いていないのでタイマ割り込みを使いソフトウェアでコントロールする必要がある。

ということになる。要するにOPNの方には肝心のFM音源が、

### 3チャンネルしか

付いていないのである。それに対してOPMには、

### 8チャンネルも

付いているのである。これがどういうことかということ、同時に演奏できる楽器の数が3対8なのである。そしてこれは数だけの問題にとどまらず、音の厚みに影響するのである。で、OPMとOPNの使い方(データ)の互換性であるが、基本的に両者には互換性がある。ただしエンベロープのタイミングなどが違い、一部に移植できないものもあるらしい。といってもVIPに付属の200音色を見ても分かるように、OPNから移植できないものがあつたとしても、別に不都合があるわけではない。

ちなみに、このFM音源ボードであるが、I/Oアドレスをずらして何枚でもスロットにさせるようにしたら非常にその筋だったと思ったのであるが、残念ながらそうはなっていないようである。

というところで、OPMの具体的な説明に入るのであった。

## OPMである

図11-1がOPMのI/Oアドレスである。テンポの制御用にZ80 CTCが付いているが、turboおよびturboZで動かす場合はこのボード上のCTCではなく、本体内蔵のCTC(アドレスは1FA0<sub>H</sub>~1FA3<sub>H</sub>)を使うことになっている。気を付けるように。



さて、OPM には 256 ( $-a$ ) 個のレジスタがあるわけだ。そして、これらのレジスタにそれなりの値を書き込むことによって、さまざまな音を出せるのである。この点は PSG (AY-3-8910) と似たようなものである。そしてレジスタに値を入れる方法であるが、たとえば 21<sub>H</sub> 番レジスタに C3<sub>H</sub> を入れるのであれば、

OUT &H700, &H21

OUT &H701, &HC3

とするのである。まったくの自然体である。

図 11-1 OPM の I/O アドレス

0700 <sub>H</sub>	YM2151 アドレスポート	OUT
0701 <sub>H</sub>	YM2151 データポート	IN/OUT
0704 <sub>H</sub>	CTC チャンネル 0	IN/OUT
0705 <sub>H</sub>	CTC チャンネル 1	IN/OUT
0706 <sub>H</sub>	CTC チャンネル 2	IN/OUT
0707 <sub>H</sub>	CTC チャンネル 3	IN/OUT

図 11-2, 11-3 がそのレジスタマップである。このマップは基本的に 00<sub>H</sub> 番～1F<sub>H</sub> 番 (途中で所々で抜けているが) と、20<sub>H</sub>～FF<sub>H</sub> の二つの部分に分かれているわけだ。アドレスの見方にちょっと注意が必要だからそのつもりで。

それで、図 11-3 を見ると分かるように、OP2 と OP3 でレジスタ番号が逆転してい

図 11-2 レジスタマップ(00<sub>H</sub>～1F<sub>H</sub>)

レジスタ番号	D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>
01 <sub>H</sub>	TEST						LFO RESET	
08 <sub>H</sub>		OPマスク				CH No.		
		OP4	OP3	OP2	OP1			
0F <sub>H</sub>	NE			NFREQ				
10 <sub>H</sub>	CLK A <sub>1</sub>							
11 <sub>H</sub>							CLK A <sub>2</sub>	
12 <sub>H</sub>	CLK B							
14 <sub>H</sub>	CMS		FRESET		IRQ EN		LOAD	
			B	A	B	A	B	A
18 <sub>H</sub>	LFRQ							
19 <sub>H</sub>	F	PMD or AMD						
1B <sub>H</sub>	CT <sub>2</sub>	CT <sub>1</sub>					W	



図 11-3 レジスタマップ(20H~FFH)

	CH0	CH1	CH2	CH3	CH4	CH5	CH6	CH7	D7	D6	D5	D4	D3	D2	D1	D0
	20 <sub>H</sub>	21 <sub>H</sub>	22 <sub>H</sub>	23 <sub>H</sub>	24 <sub>H</sub>	25 <sub>H</sub>	26 <sub>H</sub>	27 <sub>H</sub>	R	L	FEED BACK			ALG		
	28 <sub>H</sub>	29 <sub>H</sub>	2A <sub>H</sub>	2B <sub>H</sub>	2C <sub>H</sub>	2D <sub>H</sub>	2E <sub>H</sub>	2F <sub>H</sub>	KEY CODE							
	30 <sub>H</sub>	31 <sub>H</sub>	32 <sub>H</sub>	33 <sub>H</sub>	34 <sub>H</sub>	35 <sub>H</sub>	36 <sub>H</sub>	37 <sub>H</sub>	KEY FRACTION							
	38 <sub>H</sub>	39 <sub>H</sub>	3A <sub>H</sub>	3B <sub>H</sub>	3C <sub>H</sub>	3D <sub>H</sub>	3E <sub>H</sub>	3F <sub>H</sub>	PMS			AMS				
OP1	40 <sub>H</sub>	41 <sub>H</sub>	42 <sub>H</sub>	43 <sub>H</sub>	44 <sub>H</sub>	45 <sub>H</sub>	46 <sub>H</sub>	47 <sub>H</sub>				DT1		MUL		
OP2	50 <sub>H</sub>	51 <sub>H</sub>	52 <sub>H</sub>	53 <sub>H</sub>	54 <sub>H</sub>	55 <sub>H</sub>	56 <sub>H</sub>	57 <sub>H</sub>								
OP3	48 <sub>H</sub>	49 <sub>H</sub>	4A <sub>H</sub>	4B <sub>H</sub>	4C <sub>H</sub>	4D <sub>H</sub>	4E <sub>H</sub>	4F <sub>H</sub>								
OP4	58 <sub>H</sub>	59 <sub>H</sub>	5A <sub>H</sub>	5B <sub>H</sub>	5C <sub>H</sub>	5D <sub>H</sub>	5E <sub>H</sub>	5F <sub>H</sub>								
OP1	60 <sub>H</sub>	61 <sub>H</sub>	62 <sub>H</sub>	63 <sub>H</sub>	64 <sub>H</sub>	65 <sub>H</sub>	66 <sub>H</sub>	67 <sub>H</sub>				TL				
OP2	70 <sub>H</sub>	71 <sub>H</sub>	72 <sub>H</sub>	73 <sub>H</sub>	74 <sub>H</sub>	75 <sub>H</sub>	76 <sub>H</sub>	77 <sub>H</sub>								
OP3	68 <sub>H</sub>	69 <sub>H</sub>	6A <sub>H</sub>	6B <sub>H</sub>	6C <sub>H</sub>	6D <sub>H</sub>	6E <sub>H</sub>	6F <sub>H</sub>								
OP4	78 <sub>H</sub>	79 <sub>H</sub>	7A <sub>H</sub>	7B <sub>H</sub>	7C <sub>H</sub>	7D <sub>H</sub>	7E <sub>H</sub>	7F <sub>H</sub>								
OP1	80 <sub>H</sub>	81 <sub>H</sub>	82 <sub>H</sub>	83 <sub>H</sub>	84 <sub>H</sub>	85 <sub>H</sub>	86 <sub>H</sub>	87 <sub>H</sub>	KS					AR		
OP2	90 <sub>H</sub>	91 <sub>H</sub>	92 <sub>H</sub>	93 <sub>H</sub>	94 <sub>H</sub>	95 <sub>H</sub>	96 <sub>H</sub>	97 <sub>H</sub>								
OP3	88 <sub>H</sub>	89 <sub>H</sub>	8A <sub>H</sub>	8B <sub>H</sub>	8C <sub>H</sub>	8D <sub>H</sub>	8E <sub>H</sub>	8F <sub>H</sub>								
OP4	98 <sub>H</sub>	99 <sub>H</sub>	9A <sub>H</sub>	9B <sub>H</sub>	9C <sub>H</sub>	9D <sub>H</sub>	9E <sub>H</sub>	9F <sub>H</sub>								
OP1	A0 <sub>H</sub>	A1 <sub>H</sub>	A2 <sub>H</sub>	A3 <sub>H</sub>	A4 <sub>H</sub>	A5 <sub>H</sub>	A6 <sub>H</sub>	A7 <sub>H</sub>	AMS-EN					1DR		
OP2	B0 <sub>H</sub>	B1 <sub>H</sub>	B2 <sub>H</sub>	B3 <sub>H</sub>	B4 <sub>H</sub>	B5 <sub>H</sub>	B6 <sub>H</sub>	B7 <sub>H</sub>								
OP3	A8 <sub>H</sub>	A9 <sub>H</sub>	AA <sub>H</sub>	AB <sub>H</sub>	AC <sub>H</sub>	AD <sub>H</sub>	AE <sub>H</sub>	AF <sub>H</sub>								
OP4	B8 <sub>H</sub>	B9 <sub>H</sub>	BA <sub>H</sub>	BB <sub>H</sub>	BC <sub>H</sub>	BD <sub>H</sub>	BE <sub>H</sub>	BF <sub>H</sub>								
OP1	C0 <sub>H</sub>	C1 <sub>H</sub>	C2 <sub>H</sub>	C3 <sub>H</sub>	C4 <sub>H</sub>	C5 <sub>H</sub>	C6 <sub>H</sub>	C7 <sub>H</sub>	DT2					2DR		
OP2	D0 <sub>H</sub>	D1 <sub>H</sub>	D2 <sub>H</sub>	D3 <sub>H</sub>	D4 <sub>H</sub>	D5 <sub>H</sub>	D6 <sub>H</sub>	D7 <sub>H</sub>								
OP3	C8 <sub>H</sub>	C9 <sub>H</sub>	CA <sub>H</sub>	CB <sub>H</sub>	CC <sub>H</sub>	CD <sub>H</sub>	CE <sub>H</sub>	CF <sub>H</sub>								
OP4	D8 <sub>H</sub>	D9 <sub>H</sub>	DA <sub>H</sub>	DB <sub>H</sub>	DC <sub>H</sub>	DD <sub>H</sub>	DE <sub>H</sub>	DF <sub>H</sub>								
OP1	E0 <sub>H</sub>	E1 <sub>H</sub>	E2 <sub>H</sub>	E3 <sub>H</sub>	E4 <sub>H</sub>	E5 <sub>H</sub>	E6 <sub>H</sub>	E7 <sub>H</sub>	1DL			RR				
OP2	F0 <sub>H</sub>	E1 <sub>H</sub>	F2 <sub>H</sub>	F3 <sub>H</sub>	F4 <sub>H</sub>	F5 <sub>H</sub>	F6 <sub>H</sub>	F7 <sub>H</sub>								
OP3	E8 <sub>H</sub>	E9 <sub>H</sub>	EA <sub>H</sub>	EB <sub>H</sub>	EC <sub>H</sub>	ED <sub>H</sub>	EE <sub>H</sub>	EF <sub>H</sub>								
OP4	F8 <sub>H</sub>	F9 <sub>H</sub>	FA <sub>H</sub>	FB <sub>H</sub>	FC <sub>H</sub>	FD <sub>H</sub>	FE <sub>H</sub>	FF <sub>H</sub>								

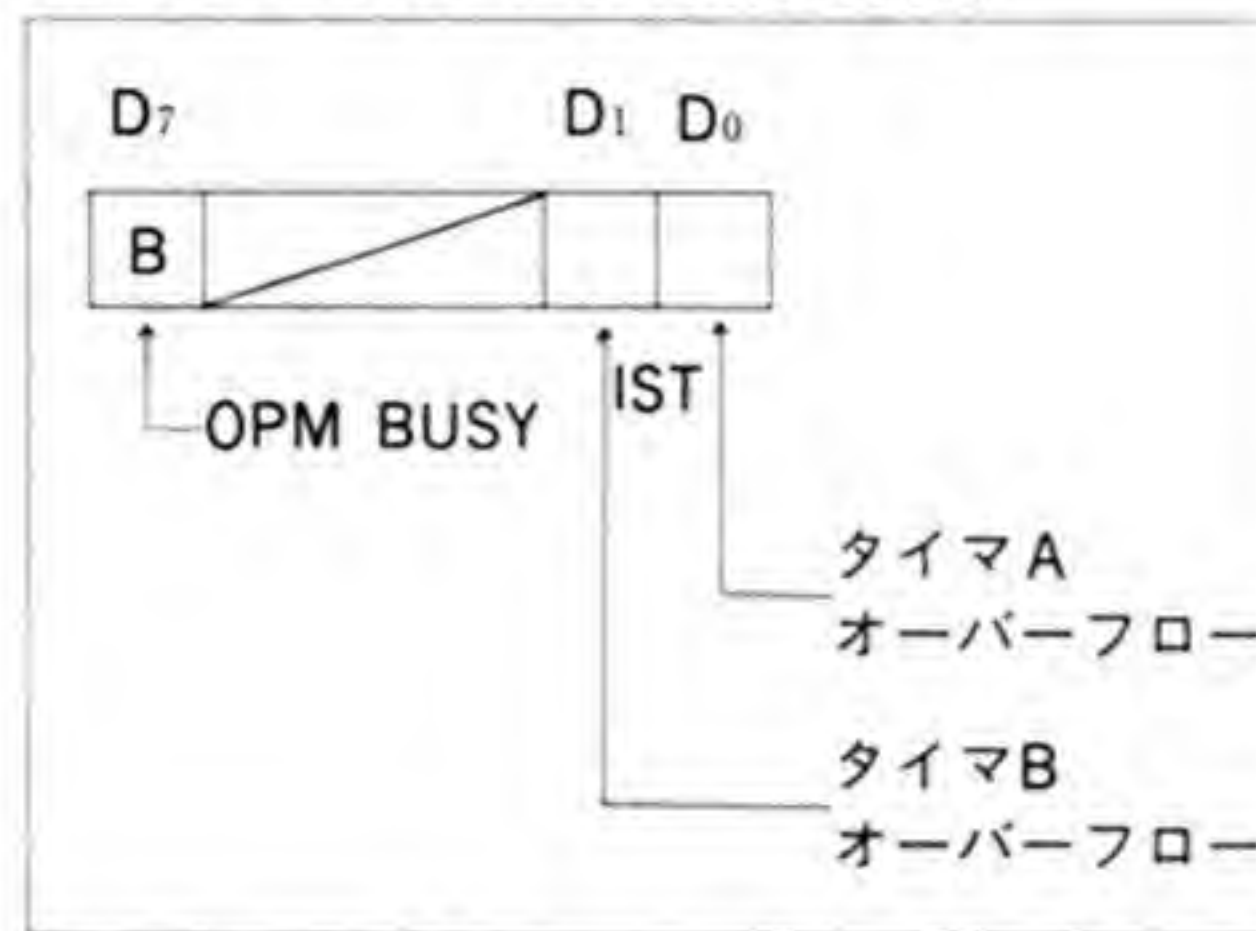
る。私は連載時にここでタコってしまってわんわんわわん、わんわんわわんだったのである。YAMAHA は一体どういう理由でこんなことをしたのであるのか？ まったく理解に苦しむのである。

ところで、「レジスタに書き込むのは分かったが、それじゃ読み出せるのか？」という疑問が湧くであろう。実はレジスタの値は読み出せないのである。その代わりと言ってはなんだが、OPM では、

INP (&H701)



図 11-4 701<sub>II</sub> からの入力の内容



とすると、図 11-4 のようなステータスが得られるのである。これにおいて、ビット 7 は「た  
だいま私こと OPM は BUSY です、データを OUT しないでください」ということな  
のである。OPM のマニュアルによると直前にデータを書き込んだ後 64 クロックの間  
BUSY になるそうである。よって本当はこのビットが 0 になっていることを確認してから  
データを送らなければならないのであるが、64 クロックといえはあつという間であるから  
よほどのことがない限り無視してよいだろう。ビット 1, 0 の IST (たぶんインタラプト・  
ステータスのことであろう) というのはオーバーフロー (割り込み) を起こした (OPM 内  
蔵の) タイマを示しているのだが、CZ-8BS1 ではこれらの割り込み機能は使えない (使う  
必要もない) ので説明はしない。どうしても知りたい人は参考文献 6 の YM 2151 のマニ  
ュアルを見ていただきたい。また、CZ-8BS1 を持っている人ならば「取扱説明書」も持って  
いるはずであるから、それも参考にしていきたい。

ではここからねっとりと解説するのであるが、ここの解説だけで OPM がきちんと理解  
できるとは思わないように。シンセサイザの世界は、また一味違ったその筋な世界らし  
いのである。

## OPMの基本

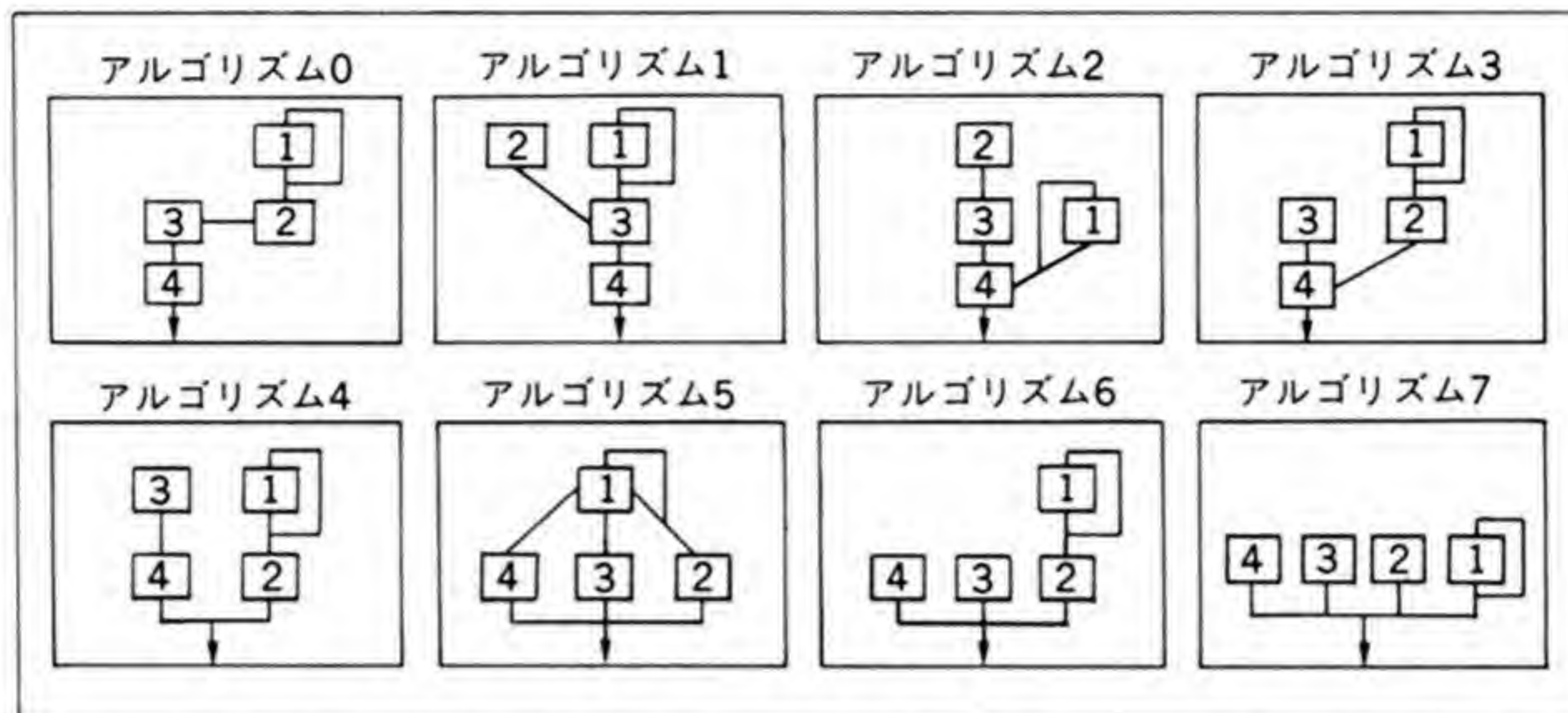
OPM には 8 チャンネル (8 ボイス) があるわけなのだが、それぞれのチャンネルはさら  
に 4 個のオペレータを組み合わせることによってでき上がっている。その四つのオペレ  
ータの組み合わせ方には 8 パターンが用意されていて、それが「アルゴリズム」と呼ばれる  
ものである。そのアルゴリズムを図 11-5 に示す。VIP (CZ-8BS1 に付属のソフト) ではど  
ういうわけか 1~8 となっているのだが、ここでは 0~7 として説明する。ところで、こ  
れら八つのアルゴリズムを見ると、1 番 (OP1) にだけ自分から出て、また自分に戻る道筋  
が書かれていることに気付くであろう。これがフィードバック (FEED BACK) というや  
つである。後で出てくるから覚えておくよーに。

オペレータとは一体いかなるものかという、基本的にこれは「サイン波発生回路」な  
のである。ただし単純にサイン波を出すだけのものではなく、あれこれといじることが可  
能で、それを利用していろいろな音を作れるのである。

さて、シンセサイザのイロハとも言えるのが図 11-6 の「エンベロープ」である (SSG :



図 11-5 アルゴリズム 8 態



AY-3-8910 のエンベロープとはちょっと違うので注意)。これは例のアタックレート、ディケイレート、サステーンレート、サステーンレベル、リリースレートというやつである。これは出力の時間変化なのである。で、先程も書いたように OPM には 8 チャンネルあり、それぞれにオペレータが 4 個ずつあるのである。つまり全部で  $8 \times 4 = 32$  個のオペレータがある。そして恐ろしいことに、これら 32 個のオペレータに、それぞれ独立にエンベロープを指定できるのである。さらにはそれぞれのオペレータの周波数を 0.5 倍～25.95 倍の間でずらしたり、さらには微妙にプラス/マイナスしたりも可能なのである。つまりはそのよーにして、微妙に周波数、出力をずらしたサイン波発生回路をあれこれと組み合わせて音を作っていくのである。そしてこれが肝心なのだが、FM 音源において「組み合わせ」というのは「足し合わせる」だけではなく、重関数も可能なのである。すなわち、

$$\text{SIN}(2\omega t + d_1) + \text{SIN}(\omega t + d_2)$$

のようなタイプだけではなく、

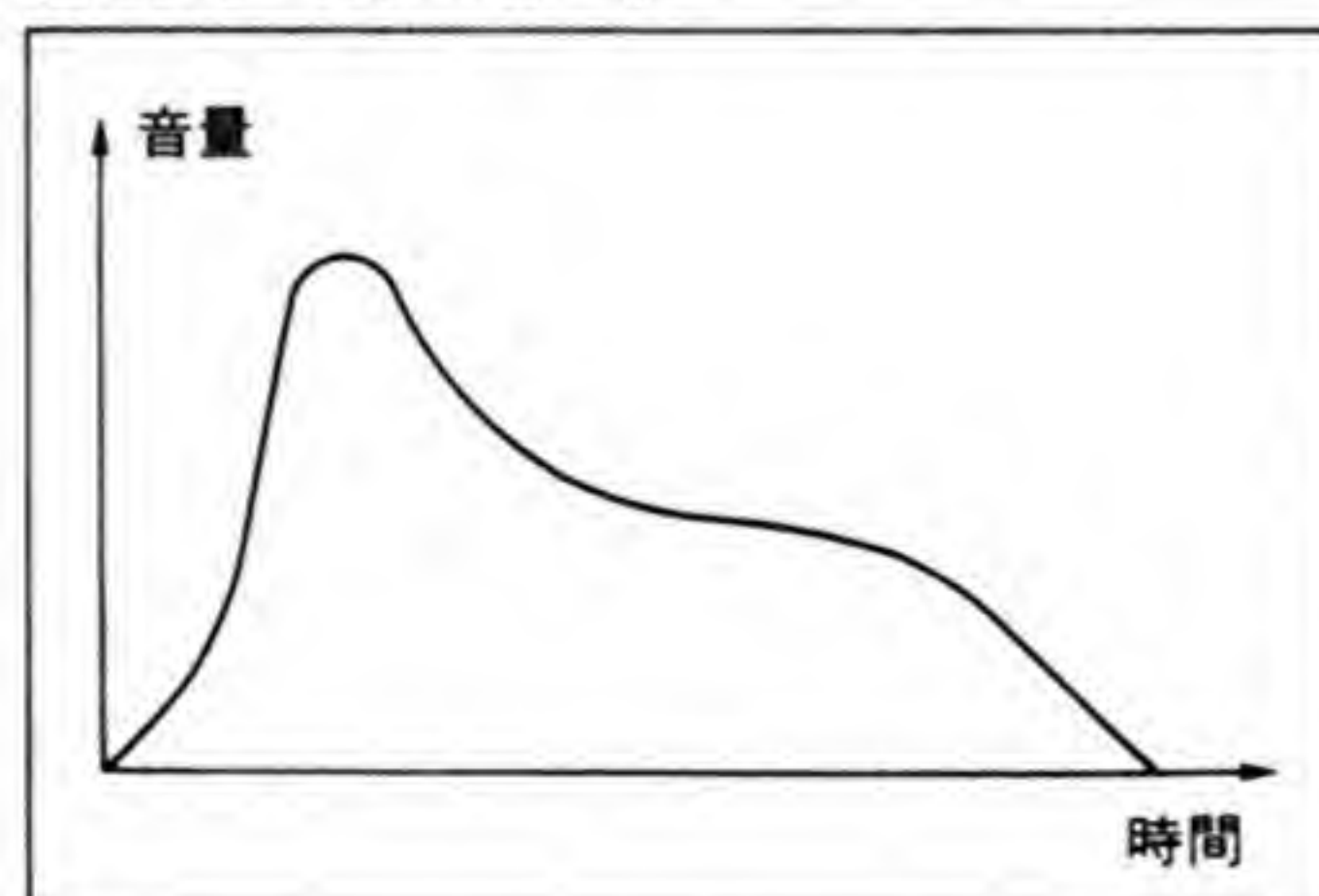
$$\text{SIN}(\omega \text{SIN}(2\omega t + d_1) + d_2)$$

などのような組み合わせができるのである。で、オペレータは 4 個もあるわけだ。さらに OP1 は自分の出力を自分に与えたりもできるのである。となると、

$$\text{SIN}(\text{SIN}(\text{SIN}(\dots) + d_1) + d_2)$$

というような波形を作ることのできるのである。となれば、「これだけ複雑なことができる

図 11-6 エンベロープ





のだから、きっといろんな音を出ることができるような気がする」であろう。ほ～ら、あなたはだんだんそんな気になってきた。ワン、ツウ、スリーである。

## OPMのレジスタである

というところで、ねっとりとレジスタの解説を始めるのである。

### ● 01<sub>H</sub> : LFO RESET (TEST)

このレジスタはもともと工場でのテスト用にあるものだが、このレジスタの第1ビットを立ち下げると（「1」→「0」）LFOのリセットが行なわれる。

LFOとは Low Frequency Oscillator=低周波発振回路のことである。これは音を微妙に震わせるために付いているのだ。音の震わせ方には、

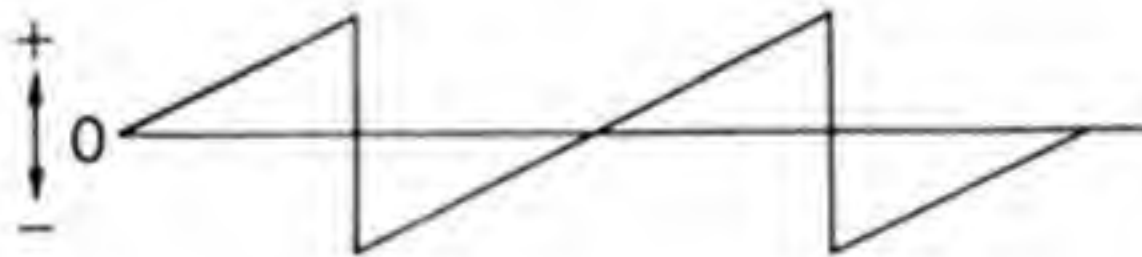
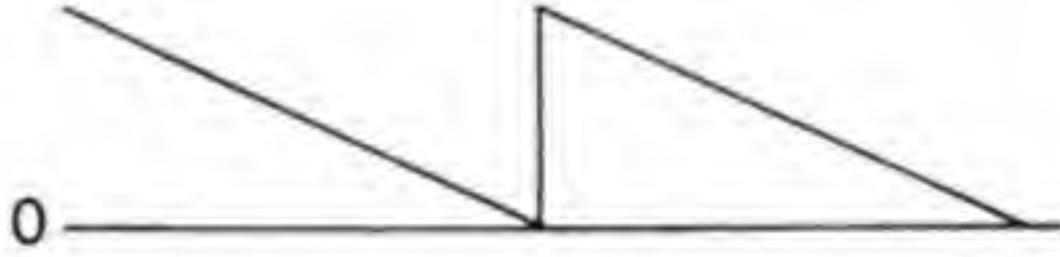
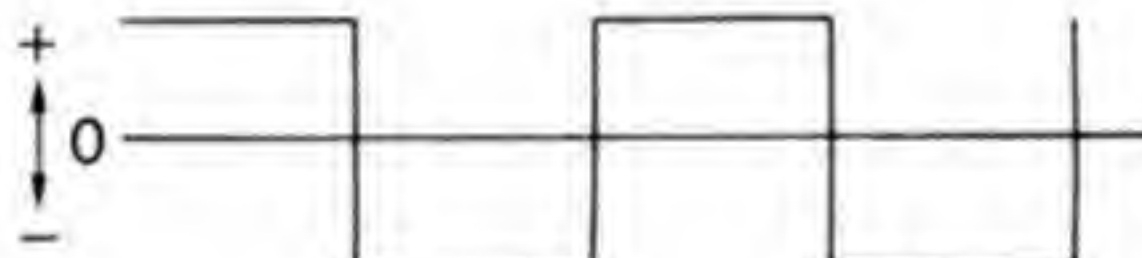
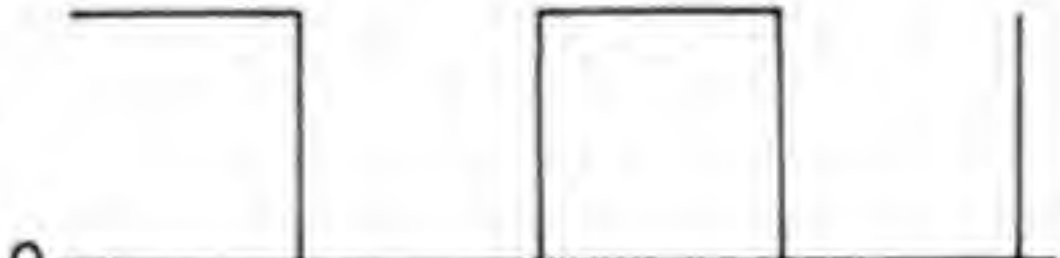
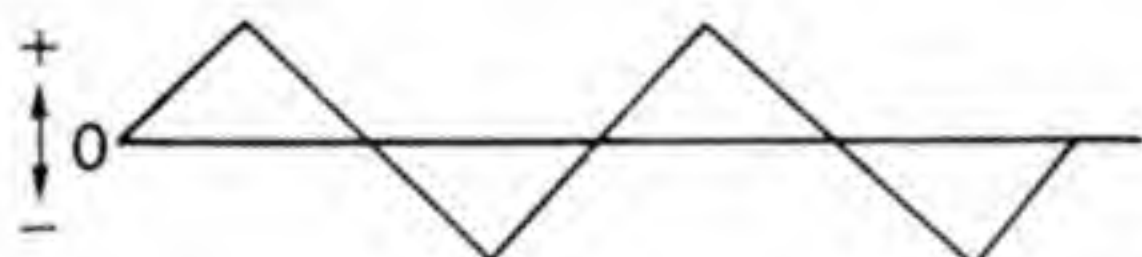
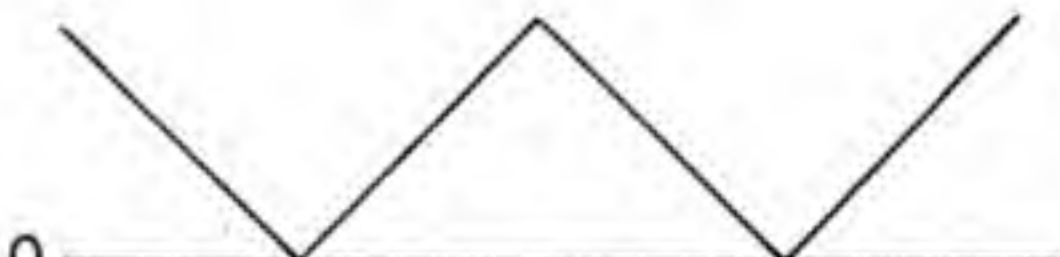
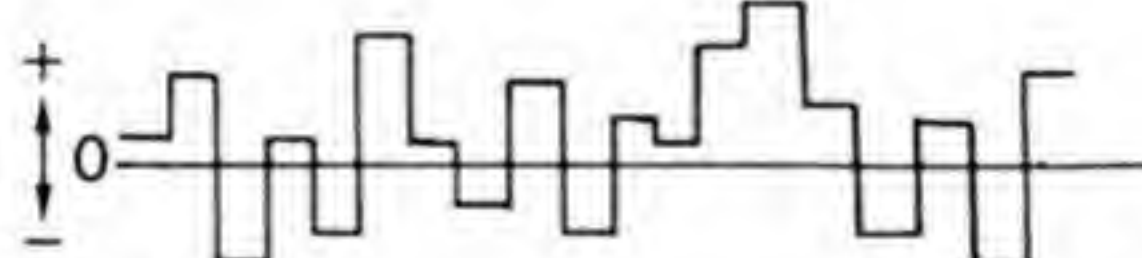
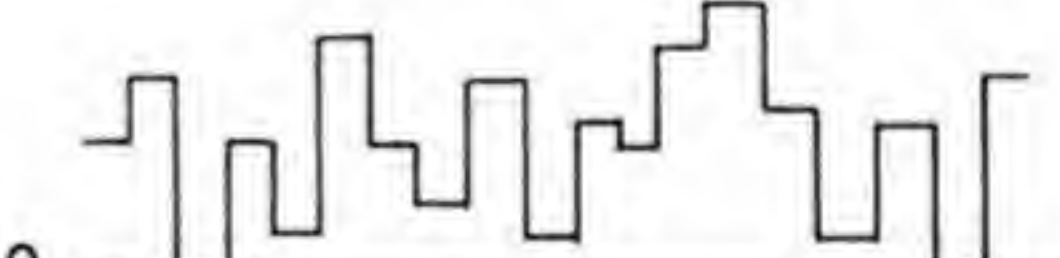
ピブラート：音程（周波数）が上下する

トレモロ：音量（音の大きさ）が上下する

の2とおりがある。LFOはこれらの「震え」の種になるものなのである。LFOには図11-7に示すような波形が用意されており、リセットとはこれらの「波形の左端から始めるようにする」ということなのである。

ちなみにSSG（AY-3-8910）の「エンベロープ」はトレモロに相当するものである。

図 11-7 LFO の波形

W	PMD (ピブラート)	AMD (トレモロ)
0 (ノコギリ波)		
1 (方形波)		
2 (三角波)		
3 (ノイズ)		

### ● 08<sub>H</sub> : KEY ON/OFF

これはそれぞれのチャンネルのそれぞれのオペレータの ON/OFF を指定するものである。たとえばこのレジスタに「1011010<sub>B</sub>」を書き込むと、下3ビットが010<sub>B</sub>=2で、その



上の4ビットが1011<sub>B</sub>だから、チャンネル2のOP4, OP2, OP1がON, OP3がOFFになるのである。たいていの場合はオペレータをすべて使うから、ここにチャンネルナンバーを書けばOFF, それに+1111000<sub>B</sub>=78<sub>H</sub>したものを書けばONということになる。

● 0F<sub>H</sub> : NE/NFREQ

OPMでは第7チャンネルのOP4だけがノイズモードを持っているのである。NE=1でノイズモードになる。NFREQはSSGのR6(ノイズ周波数)と同じようなものである。5ビット幅で大体の周波数を決めるのである。ノイズに周波数というのも変なのであるが、ま、そんなものなのである。ところでOPMではこのノイズ機能はほとんどお呼びではないのである。現実にはVIPはこの機能を使っていないのである。なぜかという、フィードバックを使うとノイズに近い音が出せてしまうからなのである。よって「チャンネル7のOP4だけ」という制限も加わって、この機能は無視する。

● 10<sub>H</sub>, 11<sub>H</sub>, 12<sub>H</sub> : CLKA, B

これはOPM内蔵のタイマの設定値である。CZ-8BS1では使わないのでいい加減に説明しておく。まずタイマAは10ビットからなる。A<sub>1</sub>が上位, A<sub>2</sub>が下位である。その10ビットの値(0~1023)をN<sub>A</sub>とすると、タイマAの周期T<sub>A</sub>は、

$$T_A = \frac{64 \times (1024 - N_A)}{f_M}$$

となる。f<sub>M</sub>はOPMに供給されているクロックで、X1では4MHzである。よってT<sub>A</sub>の最大値は16.38ms, 最小値は16μsである。

タイマBは8ビットで、

$$T_B = \frac{1024 \times (256 - CLK_B)}{f_M}$$

となっている。よって、256μs < T<sub>B</sub> < 65.536msである。

● 14<sub>H</sub> : タイマコントロール

LOADがタイマの動作開始, IRQENが割り込みを起こすかどうかのフラグ(CZ-8BS1では使えない), F RESETがオーバーフローフラグ(前述のISTである)のリセットとなっている。どれも「1」で機能する。CSM=1とするとタイマAがオーバーフローしたときにすべてのオペレータを同時にONにすることができる。ちょっと目には便利そうであるが、割り込み機能が使えないのでほとんど意味がない。ちなみになぜこの割り込み機能が使えないかというと、Z80のモード2の割り込みに対応できないからなのだ。PIOを間に入ればどうにかできるが、それならいっそのこと汎用性の高いCTCを使ってしまえということになる。

なお、オーバーフローが起きればISTの対応するビットは1になっている。

● 18<sub>H</sub> : LFRQ

LFOの周波数を決めるレジスタである。LFRQの値と実際の周波数の関係は表11-1のようになっている。この表には規則性がある、

$$f_1 = \frac{f_M \times 2^{(LFRQ/16)}}{4295 \times 10^6}$$



となっている。

表 11-1 LFO の周波数

LFRQ	周波数 (Hz)	LFRQ	周波数 (Hz)	LFRQ	周波数 (Hz)	LFRQ	周波数 (Hz)
FF	59.1278	BF	3.6955	7F	0.2310	3F	0.0144
FE	57.2205	BE	3.5763	7E	0.2235	3E	0.0140
FD	55.3131	BD	3.4571	7D	0.2161	3D	0.0135
FC	53.4058	BC	3.3379	7C	0.2086	3C	0.0130
FB	51.4984	BB	3.2187	7B	0.2012	3B	0.0126
FA	49.5911	BA	3.0994	7A	0.1937	3A	0.0121
F9	47.6837	B9	2.9802	79	0.1863	39	0.0116
F8	45.7764	B8	2.8610	78	0.1788	38	0.0112
F7	43.8690	B7	2.7418	77	0.1714	37	0.0107
F6	41.9617	B6	2.6226	76	0.1639	36	0.0102
F5	40.0543	B5	2.5034	75	0.1565	35	0.0098
F4	38.1470	B4	2.3842	74	0.1490	34	0.0093
F3	36.2396	B3	2.2650	73	0.1416	33	0.0088
F2	34.3323	B2	2.1458	72	0.1341	32	0.0084
F1	32.4249	B1	2.0266	71	0.1267	31	0.0079
F0	30.5176	B0	1.9073	70	0.1192	30	0.0075
EF	29.5639	AF	1.8477	6F	0.1155	2F	0.0072
EE	28.6102	AE	1.7881	6E	0.1118	2E	0.0070
ED	27.6566	AD	1.7285	6D	0.1080	2D	0.0068
EC	26.7029	AC	1.6689	6C	0.1043	2C	0.0065
EB	25.7492	AB	1.6093	6B	0.1006	2B	0.0063
EA	24.7955	AA	1.5497	6A	0.0969	2A	0.0061
E9	23.8419	A9	1.4901	69	0.0931	29	0.0058
E8	22.8882	A8	1.4305	68	0.0894	28	0.0056
E7	21.9345	A7	1.3709	67	0.0857	27	0.0054
E6	20.9808	A6	1.3113	66	0.0820	26	0.0051
E5	20.0272	A5	1.2517	65	0.0782	25	0.0049
E4	19.0735	A4	1.1921	64	0.0745	24	0.0047
E3	18.1198	A3	1.1325	63	0.0708	23	0.0044
E2	17.1661	A2	1.0729	62	0.0671	22	0.0042
E1	16.2125	A1	1.0133	61	0.0633	21	0.0040
E0	15.2588	A0	0.9537	60	0.0596	20	0.0037
DF	14.7820	9F	0.9239	5F	0.0577	1F	0.0036
DE	14.3051	9E	0.8941	5E	0.0559	1E	0.0035
DD	13.8283	9D	0.8643	5D	0.0540	1D	0.0034
DC	13.3514	9C	0.8345	5C	0.0522	1C	0.0033
DB	12.8746	9B	0.8047	5B	0.0503	1B	0.0031
DA	12.3978	9A	0.7749	5A	0.0484	1A	0.0030
D9	11.9209	99	0.7451	59	0.0466	19	0.0029
D8	11.4441	98	0.7153	58	0.0447	18	0.0028
D7	10.9673	97	0.6855	57	0.0428	17	0.0027
D6	10.4904	96	0.6557	56	0.0410	16	0.0026
D5	10.0136	95	0.6258	55	0.0391	15	0.0024
D4	9.5367	94	0.5960	54	0.0373	14	0.0023
D3	9.0599	93	0.5662	53	0.0354	13	0.0022
D2	8.5831	92	0.5364	52	0.0335	12	0.0021
D1	8.1062	91	0.5066	51	0.0317	11	0.0020
D0	7.6294	90	0.4768	50	0.0298	10	0.0019
CF	7.3910	8F	0.4619	4F	0.0289	0F	0.0018
CE	7.1526	8E	0.4470	4E	0.0279	0E	0.0017
CD	6.9141	8D	0.4321	4D	0.0270	0D	0.0017
CC	6.6757	8C	0.4172	4C	0.0261	0C	0.0016
CB	6.4373	8B	0.4023	4B	0.0251	0B	0.0016
CA	6.1989	8A	0.3874	4A	0.0242	0A	0.0015
C9	5.9605	89	0.3725	49	0.0233	09	0.0015
C8	5.7220	88	0.3576	48	0.0224	08	0.0014
C7	5.4836	87	0.3427	47	0.0214	07	0.0013
C6	5.2452	86	0.3278	46	0.0205	06	0.0013
C5	5.0068	85	0.3129	45	0.0196	05	0.0012
C4	4.7684	84	0.2980	44	0.0186	04	0.0012
C3	4.5300	83	0.2831	43	0.0177	03	0.0011
C2	4.2915	82	0.2682	42	0.0168	02	0.0010
C1	4.0531	81	0.2533	41	0.0158	01	0.0010
C0	3.8147	80	0.2384	40	0.0149	00	0.0009



● 19<sub>H</sub> : F, PMD or AMD

ビット 7 は指定した値が PMD 用であるか AMD 用であるかを区別するためのものである。F = 1 で PMD に、F = 0 で AMD に使われる。想像するに、OPM の内部には PMD 用と AMD 用の二つの 7 ビットの隠れワークエリアがあるのだ。そしてビット 7 = F は、ビット 0 ~ 6 の値を PMD のワークエリアの方に送るか、それとも AMD の方に送るかのフラグになっているわけである。

AMD, PMD は 0 ~ 127 の値を設定するわけだが、これは早い話が、LFO のかかり具合を決めるものである。ただし、ここだけですべて決まるものではない。詳しくは後程説明する。

ところでどーして素直に 19<sub>H</sub> と 1A<sub>H</sub> の二つのレジスタに分けなかったのだろう。不可解である。

● 1B<sub>H</sub> : CT<sub>1</sub>, CT<sub>2</sub>, W

CT<sub>1</sub>, CT<sub>2</sub> は外部をコントロールするためのものである。YM2151 には CT<sub>1</sub>, CT<sub>2</sub> というピンがあって、そこに直行しているのである。つまり 2 ビットだけの汎用出力ポートである。X1 では使えない。

W は LFO の波形を指定するものである。値と波形の関係は図 11-7 である。LFO が PMD に使われるか AMD に使われるかで少し違うことに注意。

ここから先は図 11-3 の方に差ししかかってくるわけである。8 個のチャンネルはすべて同じであるからこれから先はチャンネル 0 を対象とする。もしもチャンネル 3 を扱うのならレジスタ番号を + 3 すればよいだけ（以下同様）の話なのである。

● 20<sub>H</sub> : RL, FB, ALG

ビット 7, 6 はそのチャンネルの出力の右左を決めるものである。図 11-3 では左側になっているビット 7 が R（右出力指定）でビット 6 が L であることに注意。FB は OP1 のフィードバック（自己変調）の深さを決めるものである。0 ならば自己変調はなしである。資料には表 11-2 が載っているが、「 $\pi/8$ 」が一体何を表しているのかは、私の知ったことではない。とにかく FB を大きくするとノイズっぽい「ジャ〜ン」という音になっていく。ALG は前述のアルゴリズムを指定するものである。

表 11-2 フィードバックの深さ

FB	0	1	2	3	4	5	6	7
レベル	OFF	$\pi/16$	$\pi/8$	$\pi/4$	$\pi/2$	$\pi$	$2\pi$	$4\pi$

● 28<sub>H</sub> : KEY CODE

● 30<sub>H</sub> : KEY FRACTION

これは音階を指定するレジスタである。KEY CODE の方はドレミの音階、KEY FRACTION の方は微調整用である（半音 = 100 セントとして 1.6 セント刻みで 64 段階 = 0 ~ 100.8 セントを調整できる）。さて、OPM は実は 3.58 MHz で使うように設計されているのだが、CZ-8BS1 では 4MHz で動かしているのだ。よって音階の指定は本来のデータと違ってしまうのである。それでどうするかというと、KEY FRACTION に 5 を指定



し、後は KEY CODE を半音二つ分下げてやるのである。よって本来の 3.58 MHz では、  
 ド、レ、ミ、ファ、ソ、ラ、シ、ド は、  
 3E<sub>H</sub>, 41<sub>H</sub>, 44<sub>H</sub>, 45<sub>H</sub>, 48<sub>H</sub>, 4A<sub>H</sub>, 4D<sub>H</sub>, 4E<sub>H</sub>  
 に対応するのであるが、CZ-8BS1 では、  
 3C<sub>H</sub>, 3E<sub>H</sub>, 41<sub>H</sub>, 42<sub>H</sub>, 45<sub>H</sub>, 48<sub>H</sub>, 4A<sub>H</sub>, 4C<sub>H</sub>  
 となるのである。ドが「?C<sub>H</sub>」になったのであるから分かりやすくなったとも思えるがよう分かん。なお、KEY CODE に 10<sub>H</sub> を加えると 1 オクターブ上の音になるのである。表 11-3 を参照のこと。

表 11-3 4MHz 動作時のキーコード(KEY FRACTION=5 とすること)

KEY CODE	+0C <sub>H</sub>	+0D <sub>H</sub>	+0E <sub>H</sub>	+0F <sub>H</sub>	+10 <sub>H</sub>	+11 <sub>H</sub>	+12 <sub>H</sub>	+13 <sub>H</sub>	+14 <sub>H</sub>	+15 <sub>H</sub>	+16 <sub>H</sub>	+17 <sub>H</sub>	+18 <sub>H</sub>	+19 <sub>H</sub>	+1A <sub>H</sub>	+1B <sub>H</sub>
音 程	C	C#	D	(D#)	D#	E	F	(F#)	F#	G	G#	(A)	A	A#	B	(+C)

● 38<sub>H</sub> : PMS, AMS

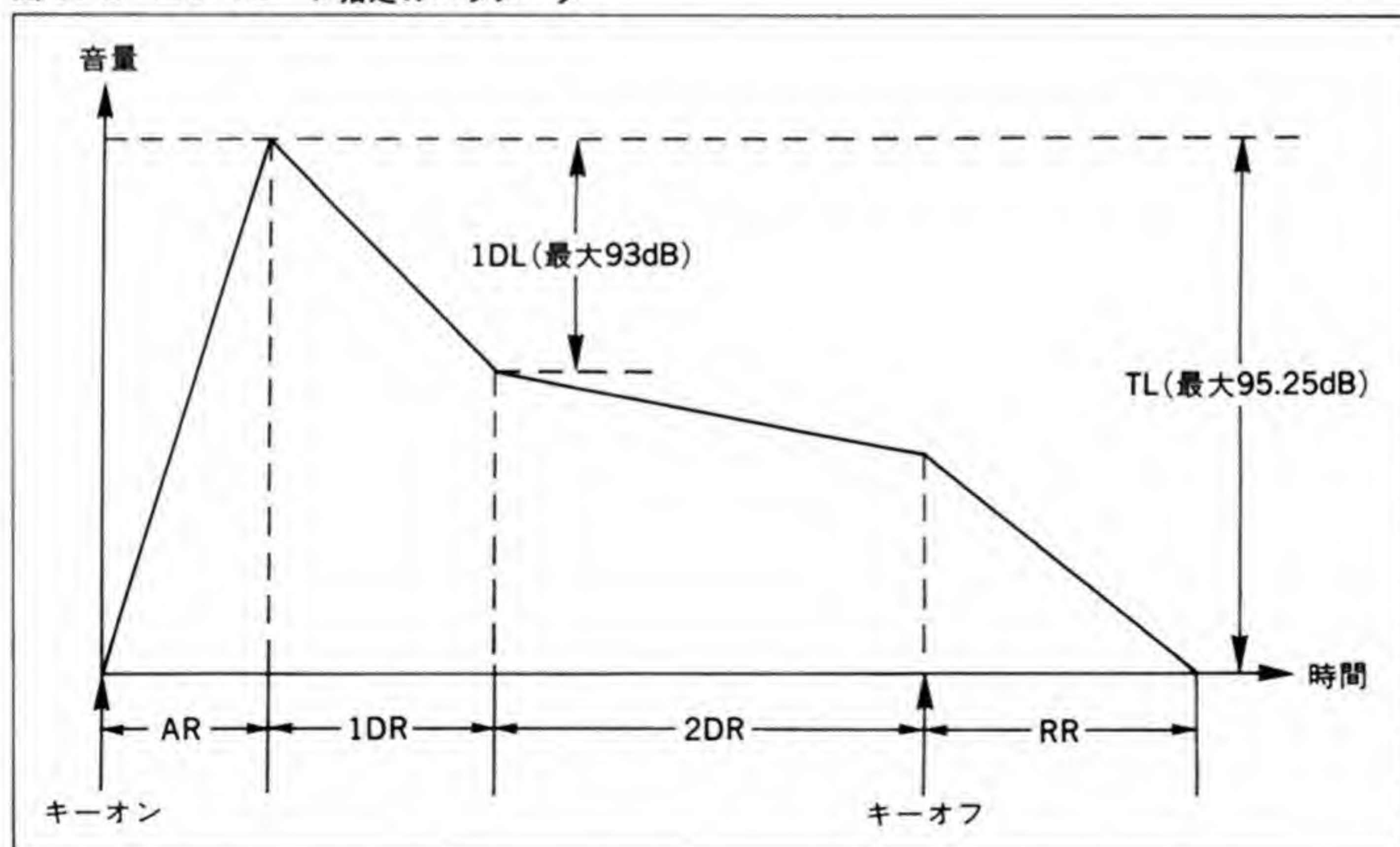
これは 19<sub>H</sub> の PMD or AMD と関係あるところである。後でまとめて解説する。

● 40<sub>H</sub> 以降はレジスタ番号順ではなく、機能ごとにまとめて説明する。

① AR, 1DR, 2DR, 1DL, RR, TL, KS

これらはエンベロープに関するパラメータである。図 11-8 に示したような機能を持っている。図からは逆のような感じがするかもしれないが、AR(アタックレート)、1DR(ファーストディケイレート)、2DR(セカンドディケイレート)、RR(リリースレート)は大きければ大きいほどエンベロープの変化が速くなるのである。1DL(ファーストディケイレベル)は少し違って、1DR → 2DR の切り換え点を示すものである。AR で最大に達した出

図 11-8 エンベロープ指定のパラメータ





力が1DRで指定された速度で1DL分減ったところで減り方が2DRに従うように切り換えられるのである。つまり、

- 1) AR: キーオンになったら、とにかくこの速さで最大の出力に達しなさい (キーオン, キーオフはレジスタ 08<sub>H</sub>で指定するんでしたねっ)。
- 2) 1DR: ARで最大に達した後はこの速さで減っていきなさい。ただし1DLで指定された分だけです。
- 3) 2DR: 1DL分減りましたね。では今からこの2DRで指定する速さで減っていきなさい。ただしキーオフになるまでです。
- 4) RR: キーオフになった後はRRで指定した速さで減りなさい。はいよくできました。

となっているのである。そして最後のTL (トータルレベル) は要するに音量なのである。TLは0で最大, 127で最小の出力となる。逆であるから注意のこと。

で、問題は「速さ」は具体的にどうなっているのかである。それは表11-4, 11-5なのであるが、これにはKS (キースケーリング) が絡んでくるのである。キースケーリングとはどういうものかというと、

**音の高さに応じてエンベロープの変化も速くする** (そういうモードも付けておこう)

表 11-4 KS, KC' → R<sub>KS</sub>

KC' \ KS	0	1	2	3	KC' \ KS	0	1	2	3
0	0	0	0	0	16	2	4	8	16
1	0	0	0	1	17	2	4	8	17
2	0	0	1	2	18	2	4	9	18
3	0	0	1	3	19	2	4	9	19
4	0	1	2	4	20	2	5	10	20
5	0	1	2	5	21	2	5	10	21
6	0	1	3	6	22	2	5	11	22
7	0	1	3	7	23	2	5	11	23
8	1	2	4	8	24	3	6	12	24
9	1	2	4	9	25	3	6	12	25
10	1	2	5	10	26	3	6	13	26
11	1	2	5	11	27	3	6	13	27
12	1	3	6	12	28	3	7	14	28
13	1	3	6	13	29	3	7	14	29
14	1	3	7	14	30	3	7	15	30
15	1	3	7	15	31	3	7	15	31

表 11-5 出力変化時間表

A 10% ↔ 90%

ATTACK TIME		DECAY TIME	
RATE	ms(10%→90%)	RATE	ms(90%→10%)
63	0.00	63	1.22
62	0.24	62	1.22
61	0.24	61	1.22
60	0.24	60	1.22
59	0.30	59	1.39

B 0% ↔ 100%

ATTACK TIME		DECAY TIME	
RATE	ms(0%→100%)	RATE	ms(100%→0%)
63	0.00	63	6.02
62	0.47	62	6.02
61	0.47	61	6.02
60	0.47	60	6.02
59	0.57	59	8.03



58	0.36	58	1.62	58	0.67	58	8.03
57	0.42	57	1.95	57	0.81	57	9.63
56	0.59	56	2.43	56	1.00	56	12.04
55	0.55	55	2.78	55	1.09	55	13.77
54	0.65	54	3.26	54	1.27	54	16.06
53	0.78	53	3.89	53	1.53	53	19.27
52	0.98	52	4.87	52	1.91	52	24.08
51	1.12	51	5.57	51	1.99	51	27.52
50	1.31	50	6.49	50	2.33	50	32.11
49	1.57	49	7.79	49	2.78	49	38.53
48	1.96	48	9.74	48	3.48	48	48.16
47	2.24	47	11.12	47	3.98	47	55.04
46	2.61	46	12.99	46	4.65	46	64.22
45	3.13	45	15.58	45	5.58	45	77.06
44	3.91	44	19.48	44	6.97	44	96.33
43	4.48	43	22.26	43	7.97	43	110.09
42	5.22	42	25.96	42	9.29	42	128.43
41	6.27	41	31.16	41	11.15	41	154.12
40	7.87	40	38.95	40	13.94	40	192.65
39	8.95	39	44.52	39	15.93	39	220.17
38	10.44	38	52.83	38	18.58	38	212.12
37	12.52	37	62.32	37	22.30	37	308.25
36	15.65	36	77.90	36	27.88	36	385.31
35	17.89	35	89.03	35	31.86	35	440.35
34	20.87	34	103.86	34	41.53	34	513.74
33	25.05	33	124.64	33	44.60	33	616.48
32	31.32	32	155.80	32	55.75	32	770.60
31	35.78	31	177.43	31	63.72	31	835.95
30	41.75	30	207.74	30	74.34	30	1027.48
29	50.10	29	249.28	29	89.20	29	1232.97
28	62.62	28	311.60	28	111.51	28	1541.22
27	71.57	27	356.12	27	127.43	27	1761.39
26	83.50	26	415.47	26	148.68	26	2296.04
25	100.20	25	498.57	25	178.41	25	2465.94
24	125.26	24	623.21	24	223.01	24	3082.42
23	143.15	23	712.23	23	254.87	23	3522.77
22	167.01	22	830.94	22	297.35	22	4109.90
21	200.40	21	997.13	21	356.82	21	4931.89
20	250.50	20	1246.41	20	446.02	20	6164.86
19	286.29	19	1424.47	19	509.74	19	7045.55
18	334.01	18	1661.88	18	594.69	18	8228.76
17	400.81	17	1994.26	17	713.63	17	9863.77
16	501.01	16	2492.83	16	892.04	16	12329.71
15	572.59	15	2848.95	15	1019.48	15	14091.09
14	668.01	14	3323.77	14	1189.38	14	16439.61
13	801.62	13	3988.52	13	1427.27	13	19727.54
12	1002.02	12	4985.65	12	1784.08	12	24659.42
11	1145.16	11	5697.88	11	2038.95	11	28182.20
10	1336.02	10	6647.53	10	2378.78	10	32879.23
9	1603.23	9	7977.05	9	2854.53	9	39455.07
8	2004.04	8	9971.30	8	3568.16	8	49318.84
7	2290.32	7	11395.78	7	4077.90	7	56364.40
6	2672.05	6	13295.07	6	4757.55	6	65758.46
5	3206.45	5	15954.08	5	5709.06	5	78910.15
4	4008.07	4	19942.60	4	7136.33	4	98637.69
3	無限大	3	無限大	3	無限大	3	無限大
2	無限大	2	無限大	2	無限大	2	無限大
1	無限大	1	無限大	1	無限大	1	無限大
0	無限大	0	無限大	0	無限大	0	無限大



というものである。保証の限りではないがそうした方がより自然な音色になるそうなのである。ではキースケーリングを説明する。

まずは R を求める。AR, 1DR, 2DR の場合ならばそれらの値（レジスタに書き込んだ値）が R そのものであるが、そうではなくて、もしも RR の場合ならば、

$$R = 2 \times RR + 1$$

とするのである。

次に表 11-4 を使って  $R_{KS}$  を求める。横のカギが KS (0 ~ 3), 縦のカギが KEY CODE の上位 5 ビット =  $KC'$  (0 ~ 31) である。たとえば KS = 2, KEY CODE =  $2E_H = 0101110_R$  とすると,  $KC' = 01011_B = 11$  となるので表より  $R_{KS}$  は 5 となる。

R と  $R_{KS}$  が求められたならば、

$$RATE = \text{MAX}(2 \times R + R_{KS}, 63)$$

とする。つまり、RATE の上限を 63 に制限するのである。この RATE を使い、表 11-5 A, 11-5 B から時間を計算することになる。表 11-5 A の方は出力が 10 % → 90 % の変化をする時間、表 11-5 B の方は 0 % → 100 % の変化をする時間となっている。で、両方の表とも左側はアタックレート (AR) によって増加する時間、右側はディケイレート (1DR, 2DR, RR) によって減少する時間である。単位はすべて ms (1/1000 秒) となっている。

あと説明が必要なのが 1DL であるが、これは、

$$\text{減衰量} = 1DL \times 3(\text{dB})$$

となっている。ただし、1DL が最大の 15 のときはさらに +48 dB することになっている（つまり最大減衰量は  $15 \times 3 + 48 = 93 \text{ dB}$ ）。dB とは減衰量を表す単位で、デシベルと読む。ベルとは電話を発明したグラハム・ベルに由来した単位である。TL の方は、

$$\text{減衰量} = TL \times 0.75(\text{dB})$$

となっており、これの最大減衰量は 95.25 dB である。大体の目安となるであろう。

## ② DT1, DT2, MUL

これら KEY CODE と KEY FRACTION で決めた周波数を微調節するパラメータである。KEY CODE と KEY FRACTION は各チャンネルに一つだが、DT1, DT2, MUL は各オペレータごとに設定が可能である。すなわち一つのチャンネルでオペレータが別々の周波数で動作するのだ。まずは表 11-6 である。横のカギが MUL (0 ~ 15), 縦のカギが DT2 (0 ~ 3) である。この表から得られた値を Fr (FREQUENCY RATIO) とする。

表 11-6. DT2, MUL → Fr 表

DT2 \ MUL	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0 → 1 倍	0.50	1.00	2.00	3.00	4.00	5.00	6.00	7.00	8.00	9.00	10.00	11.00	12.00	13.00	14.00	15.00
1 → $\sqrt{2}$ 倍	0.71	1.41	2.82	4.24	5.65	7.07	8.46	9.89	11.30	12.72	14.10	15.55	16.96	18.37	19.78	21.20
2 → $\sqrt{2.5}$ 倍	0.78	1.57	3.14	4.71	6.28	7.85	9.42	10.99	12.56	14.13	15.70	17.27	18.84	20.41	21.98	23.55
3 → $\sqrt{3}$ 倍	0.87	1.73	3.46	5.19	6.92	8.65	10.38	12.11	13.84	15.57	17.30	19.03	20.76	22.49	24.22	25.95



表 11-7 KC', DT1→Fd 表

DT1 KC'	セント(半音 = 100 セント)				Hz			
	0	± 1	± 2	± 3	0	± 1	± 2	± 3
0	0.000	0.000	5.025	10.036	0.000	0.000	0.053	0.107
1	0.000	0.000	4.228	8.445	0.000	0.000	0.053	0.107
2	0.000	0.000	3.559	7.110	0.000	0.000	0.053	0.107
3	0.000	0.000	2.993	5.980	0.000	0.000	0.053	0.107
4	0.000	2.515	5.025	5.025	0.000	0.053	0.107	0.107
5	0.000	2.115	4.228	6.338	0.000	0.053	0.107	0.160
6	0.000	1.778	3.555	5.330	0.000	0.053	0.107	0.160
7	0.000	1.496	2.990	4.483	0.000	0.053	0.107	0.160
8	0.000	1.258	2.515	5.025	0.000	0.053	0.107	0.213
9	0.000	1.057	3.170	4.225	0.000	0.053	0.160	0.213
10	0.000	0.889	2.667	3.555	0.000	0.053	0.160	0.213
11	0.000	0.748	2.242	3.735	0.000	0.053	0.160	0.267
12	0.000	1.258	2.515	3.143	0.000	0.107	0.213	0.267
13	0.000	1.057	2.114	3.170	0.000	0.107	0.213	0.320
14	0.000	0.889	1.778	2.667	0.000	0.107	0.213	0.320
15	0.000	0.748	1.869	2.615	0.000	0.107	0.267	0.373
16	0.000	0.629	1.572	2.515	0.000	0.107	0.267	0.427
17	0.000	0.793	1.566	2.114	0.000	0.160	0.320	0.427
18	0.000	0.667	1.334	2.001	0.000	0.160	0.320	0.480
19	0.000	0.561	1.308	1.869	0.000	0.160	0.373	0.533
20	0.000	0.629	1.258	1.729	0.000	0.213	0.427	0.587
21	0.000	0.529	1.057	1.566	0.000	0.213	0.427	0.640
22	0.000	0.445	1.001	1.445	0.000	0.213	0.480	0.693
23	0.000	0.467	0.935	1.308	0.000	0.267	0.533	0.747
24	0.000	0.393	0.865	1.258	0.000	0.267	0.587	0.853
25	0.000	0.397	0.793	1.123	0.000	0.320	0.640	0.907
26	0.000	0.334	0.723	1.056	0.000	0.320	0.693	1.013
27	0.000	0.327	0.654	0.935	0.000	0.373	0.747	1.067
28	0.000	0.315	0.629	0.865	0.000	0.427	0.853	1.173
29	0.000	0.315	0.629	0.865	0.000	0.427	0.853	1.173
30	0.000	0.315	0.629	0.865	0.000	0.727	0.853	1.173
31	0.000	0.315	0.629	0.865	0.000	0.427	0.853	1.173

次に DT1 (−3 ~ +3) と、先程出てきた KEY CODE の上位 5 ビット分の値, KC' を使って表 11-7 から Fd を得る。DT1 はレジスタ 40<sub>H</sub> ~ 5F<sub>H</sub> の第 6, 5, 4 ビットであるが、第 6 ビットを符号ビットと解釈するのである。そして Fd と DT1 の符号は同じにすること。つまり KC' = 11, DT1 = −2 であったなら, Fd = −0.160(Hz) = −2.242(セント) となる。

以上のように Fr と Fd が求められたなら, KEY CODE と KEY FRACTION (+LFO による震え) で決まっていた周波数 F に対して,

$$F' = F \times Fr + Fd$$

としてこの F' を各オペレータの周波数とするのである。繰り返すが F' は各オペレータごとに変わることができるのである。

### ③ PMD, AMD, PMS, AMS, AMD-EN

いよいよ最後である。これは図 11-9 を見ていただくと一目瞭然であろう。そして, PMD or AMD が最大(つまり 127) のときの PMS のかかり具合は表 11-8 である。PMS = 7 の



図 11-9 パラメータの流れ

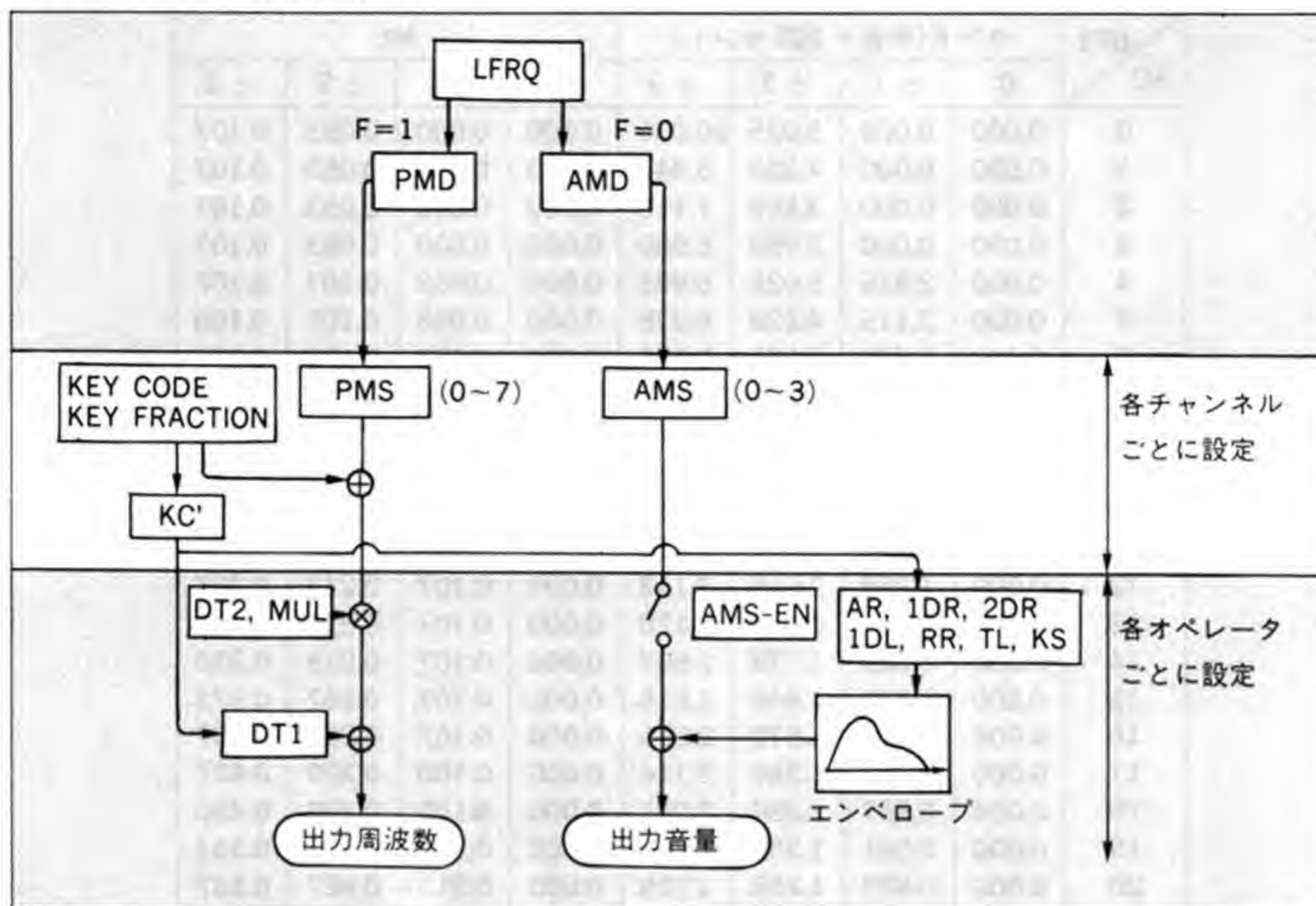


表 11-8 PMS による変調度(PMD=127 のとき)

PMS	0	1	2	3	4	5	6	7
最大変調度(セント) (半音 = 100 セント)	0	± 5	± 10	± 20	± 50	± 100	± 400	± 700

ときは 700 セントであるから、つまりは半音 7 個分なのである。AMD の方は、減衰量は  $AMD=0 \rightarrow 0\text{dB}$ ,  $1 \rightarrow 23.90625\text{dB}$ ,  $2 \rightarrow 47.8125\text{dB}$ ,  $3 \rightarrow 95.625\text{dB}$  となっている。TL による最大減衰量が 95.25dB であるから、要するに AMD を最大限にかけると谷底で出力は 0 になってしまうのである。

というところで OPM のレジスタの説明は終わりである。これだけではわけが分からないかもしれないので、ちょっとサンプルを載せておく。リスト 11-1 が VIP の NEWTONE から持ってきたピアノの音色データである。リスト 11-2 の 231~232 行を挿入して、480 行以降を書き換えるとドミソの和音となる。なおデータ文中の数値は 2 桁ならば 16 進数、1 桁もしくは 3 桁ならば 10 進数、それ以外ならば 2 進数と解釈されるようになっている。データ分を適当に書き換えて OPM をいじりまわしていただきたい。

リスト 11-1 ピアノ

```

100 READ R$:R1$=LEFT$(R$,1):IF R1$="!" THEN END
110 IF R1$="P" THEN PAUSE VAL(MID$(R$,2,255)):GOTO 100
120 R=VAL("&H"+R$)
130 READ D$:DL=INSTR(D$," ")
140 IF DL=0 THEN DL=LEN(D$)+1

```



```

150 ON DL GOTO 170,170,180,170
160 E$="&B":GOTO 190
170 E$="":GOTO 190
180 E$="&H"
190 D=VAL(E$+D$):'IF D<>0 THEN PRINTHEX$(R),D
200 GOSUB"SFM":GOTO 100
210 '
220 LABEL"SFM"
230 'WHILE(INP(&H701) AND &H80):WEND
240 OUT &H700,R
250 OUT &H701,D
260 RETURN
270 '
280 DATA 01,02,01,00 : 'TEST
290 DATA 08,00,08,01,08,02,08,03: 'KEY OFF (0-3)
300 DATA 08,04,08,05,08,06,08,07: 'KEY OFF (4-7)
310 DATA 0F,00000000 : 'NE/FREQ
320 DATA 10,00,11,00,12,00,14,00: 'CLK A1,A2,B
330 DATA 18,220 : 'LFRQ
340 DATA 19,00000100 : 'PMD/AMD
350 DATA 1B,00000010 : 'CT/W
360 DATA 20,11111010 : 'RL/FL/CON
370 DATA 30,00010100 : 'KF
380 DATA 38,00010001 : 'PMS/AMS
390 DATA 40,01010001 : 'DT1/MUL(1)
400 DATA 50,01110001 : 'DT1/MUL(2)
410 DATA 48,00100101 : 'DT1/MUL(3)
420 DATA 58,00010001 : 'DT1/MUL(4)
430 DATA 60,037,70,062,68,077,78,010: 'TL(1-4)
440 DATA 80,5F,90,56,88,5D,98,9F: 'KS/AR(1-4)
450 DATA A0,05,B0,00,A8,00,B8,87: 'AMS-EN/D1R(1-4)
460 DATA C0,07,D0,04,C8,04,D8,06: 'DT2/D2R(4)
470 DATA E0,94,F0,45,E8,45,F8,45: 'D1L/RR(4)
480 DATA 28,3C,08,78,P9,08,00,P6
490 DATA 28,3E,08,78,P9,08,00,P6
500 DATA 28,41,08,78,P9,08,00,P6
510 DATA 28,42,08,78,P9,08,00,P6
520 DATA 28,45,08,78,P9,08,00,P6
530 DATA 28,48,08,78,P9,08,00,P6
540 DATA 28,4A,08,78,P9,08,00,P6
550 DATA 28,4C,08,78,P9,08,00,P6
560 DATA !

```

リスト 11-2 ドミソ用変更点

```

231 IF (&H28<=R) AND (R<=&H2F) THEN 240
232 IF (R>=&H20) THEN FOR L=0 TO 2:GOSUB240:R=R+1:NEXT:RETURN

480 DATA 28,3C,29,41,2A,45
490 DATA 08,78,P4,08,79,P4,08,7A,P4
500 DATA 08,00,08,01,08,02
510 DATA 08,78,08,79,08,7A,P9
520 DATA 08,00,08,01,08,02,P6
530 DATA !

```

## トーンダイアラである

これから MML の作成に取りかかるわけであるが、その前に FM 音源の、なかなかその筋な使い方を発見したので紹介しておくのである。それは FM 音源で電話をかけてしまおうというものである。つまりトーンダイアラなわけだ。ただしこれはプッシュホンだけ



に有効なワザである（世の中には一見するとプッシュホンだが、実はダイヤル式の電話を押しボタン式に変えただけのものもある。そのような方式だと使えない）。

解説しよう。プッシュホンは、ボタンを押すとそれに対応した2種類の高さの正弦波が出るような仕組みになっているのである。例の「ピポパピポ」という音である。で、具体的にどのような周波数の音が出ているかというところが図11-10である。一目瞭然であろうが、たとえば「5」のボタンを押すと770Hzと1336Hzの二つの正弦波が出るのである。ちなみにみんな知っているように、一般の電話機にはA、B、C、Dのボタンは付いていない。しかしこのように決まっているそうである。

図 11-10 押しボタンの配列と周波数の関係

高群 低群				
	1209Hz	1336Hz	1477Hz	1633Hz
697Hz	1	2	3	A
770Hz	4	5	6	B
852Hz	7	8	9	C
941Hz	*	0	#	D

さて、ここからが肝心なのであるが、実はプッシュホンでボタンを押して出る「ピポパピポ」の音も、人間が送話口でしゃべる音も、電話回線にとっては同じなのである。よって、正確な周波数で口笛を吹ける人が2人いて、電話機の送話口に向かって「ピポパピ」と口笛を吹くと、ちゃんと電話がかかってしまうのである。しかし残念なことにそこまで正確に口笛を吹ける人を2人も連れてくることは難しいので、その代わりにFM音源を使ってしまおうというわけである。そしてそれがリスト11-3である。使い方は、

- 1) 正確に打ち込む。
- 2) 110行を取り去り、120行のREM(')を消してRUNする。
- 3) キーボードの数字キーを押し、プッシュホンのボタンを押したときに聞こえる音と、できるだけ近くなるようにボードに付いているボリュームを調節する。
- 4) 電話の受話器を取り、送話口をスピーカに近づけ、ピポパする。ただしこのとき117などのように緊急電話に近い番号は避ける。最初は自分の電話番号にかけて、話し中であることを確認するのがよいであろう。もしも間違い電話になってしまったらよく謝ること。

というわけで、この機能を使えばソフトだけで（送話口をスピーカに持ってくる必要はあるが）オートダイヤルできるのである。住所録や電話帳ソフトに期待したい。なお、以上の行動はNTTから怒られることはないはずであるから安心して使っていただきたい



(だってモジュージャックに直接接続するわけではない)。

ちなみにこのテクニックを応用すれば、ソフトウェアだけで「送信だけの音響カプラ」ができるはずである(きっと信頼性は最低だろう)。ぜひとも自由研究していただきたい。

### リスト 11-3 トーンダイアラ

```
100 GOSUB "INIT"
110 D$="604 1111":GOSUB 150
120 'D$=INKEY$(1):GOSUB 150:GOTO120
130 END
140 '
150 FOR I=1 TO LEN(D$)
160   A$=MID$(D$,I,1):PRINT A$;
170   GOSUB200
180 NEXT:RETURN
190 '
200 T=INSTR("123456789*0#",A$):IF T=0 THEN RETURN
210 T=T-1:L=T ¥ 3:H=4+(T MOD 3)
220 R=8:D=&H78+L:GOSUB 380
230 R=8:D=&H78+H:GOSUB 380
240 PAUSE 1
250 R=8:D=L:GOSUB 380
260 R=8:D=H:GOSUB 380
270 RETURN
280 '
290 LABEL "INIT"
300 READ R$:R1$=LEFT$(R$,1):IF R1$="!" THEN RETURN
310 R=VAL("&H"+MID$(R$,1,2))
320 M$=MID$(R$,3,1):IF M$<>"-" THEN 340
330 R0=R:R1=VAL("&H"+MID$(R$,4,2))
340 READ D$:D=VAL("&H"+D$)
350 GOSUB 370:GOTO 300
360 '
370 IF M$="-" THEN FOR R=R0 TO R1:GOSUB380:NEXT:RETURN
380 OUT &H700,R
390 OUT &H701,D
400 RETURN
410 '
420 DATA 01,02,01,00      :'TEST
430 DATA 08,00,08,01,08,02,08,03:'KEY OFF (0-3)
440 DATA 08,04,08,05,08,06,08,07:'KEY OFF (4-7)
450 DATA 0F-1B,00        :'NE/FREQ...
460 DATA 20-27,47        :'RL/FL/CON
470 DATA 28,51,29,54,2A,56,2B,59:'KCODE (0-3)
480 DATA 2C,5E,2D,61,2E,62,2F,00:'KCODE (4-7)
490 DATA 30,EC,31,A8,32,6C,33,24:'KF (0-3)
500 DATA 34,78,35,34,36,EC,37,00:'KF (4-7)
510 DATA 38-3F,00:'PMS/AMS
520 DATA 40-5F,01        :'DT1/MUL
530 DATA 60-7F,20        :'TL
540 DATA 80-9F,1F        :'KS/AR
550 DATA A0-DF,00        :'AMS-EN/1DR,DT2/2DR
560 DATA E0-FF,FF        :'1DL/RR
570 DATA !
```

というところで、FM 音源の具体的な応用として MML (ミュージック・マクロ・ランゲージ: よーするに BASIC の PLAY 文が多少進化したやつ) を作ったりするわけだ。そしてあな恐ろしや、なんと BASIC の書き換えに手を染めてしまったのであった。そう、はっきり言って私は BASIC の書き換えが嫌いなのである。なぜかという、最大の理由は BASIC というものは結構ちまちまとバグ取りのためにバージョンアップされていたりするからである。大抵の場合はそれでもたいした影響はないのであるが、しかしやはりそこ



## MMLである

さて使い方である。まずは NEW ON &HC000 を実行した後でリスト 11-5 を打ち込み、チェックサムと CRC を確認する（付録 A を使うこと）。X1 ユーザーであればここまでである。もしも turbo ユーザーである場合は、さらに次の 3 か所を変更する。

A9E8 : 07 07 → A3 1F

次に使用方法である。CZ-8FB01 V1.0 を起動する。NEW BASIC ではないし, turbo BASIC でもない。NEW ON &HB000 を実行した後, LOADM "MML. OBJ" でロードし, CALL &HA8B0 を実行。これで MML が使えるようになる。

```

                                .Z80
                                .PHASE 0A8B0H
                                ;
0704      CTC      EQU      0704H
4000      WTOP     EQU      4000H
                                ;
A8B0      21 A900      LD      HL,0A900H
A8B3      22 2DE3      LD      (2DE3H),HL
A8B6      22 2E11      LD      (2E11H),HL
A8B9      22 2E13      LD      (2E13H),HL
                                ;
A8BC      C9           RET
                                ;
A8BD      KOKO:      DS      0A900H-KOKO
                                ;
A900      CD 7FD1      CALL    7FD1H
A903      3A A5DB      LD      A,(0A5DBH)      ;A=VAR TYPE
A906      E5           PUSH    HL              ;SAVE TEXT PTR
A907      EB           EX      DE,HL
A908      FE 03        CP      03              ;STRING ?
A90A      28 72        JR      Z,DOSTR        ;GET STRING
                                ;
                                ;ELSE TEMPO
                                ;===== TEMPO =====
A90C      CD 5436      STEMP0: CALL    5436H      ;CSNG
A90F      CD 5BAF      CALL    5BAFH      ;->INT (->HL)
A912      7C           LD      A,H
A913      B5           OR      L
A914      20 4C        JR      NZ,PRET
                                ;TEMPO 0=INITIALIZE
A916      21 4001      INIT:   LD      HL,WTOP+1
A919      22 A9D4      LD      (HEAD),HL
A91C      22 A9D6      LD      (HEAD0),HL
A91F      2B           DEC     HL
A920      22 A9D2      LD      (TAIL),HL      ;INIT POINTERS
A923      44           LD      B,H
A924      4D           LD      C,L            ;BC=HL

```



A925	AF	XOR	A	
A926	ED 79	OUT	(C),A	;DUMMY DATA
A928	3E 1E	LD	A,30	;TEMPO 120
A92A	32 AE17	LD	(TMPV),A	;TEMPO AREA
;				
A92D	21 AE78	LD	HL,DFLDMY	;INIT DFLW
A930	11 AE7C	LD	DE,DFLW	
A933	01 0024	LD	BC,4*9	
A936	ED B0	LDIR		
A938	23	INC	HL	
A939	36 0C	LD	(HL),12	
A93B	2B	DEC	HL	
A93C	01 0008	LD	BC,4*2	
A93F	ED B0	LDIR		
;DE=ACTF				
A941	EB	EX	DE,HL	;HL=ACTF
A942	06 0B	LD	B,11	
A944	36 00	INIT0: LD	(HL),0	;KILL
A946	23	INC	HL	
A947	10 FB	DJNZ	INIT0	
;				
A949	F3	DI		
A94A	3E C3	LD	A,0C3H	
A94C	32 013C	LD	(13CH),A	
A94F	21 A964	LD	HL,QUIET1	
A952	22 013D	LD	(13DH),HL	
A955	CD 013C	CALL	013CH	;QUIET
;				
A958	CD A9D8	CALL	TOCTC	
A95B	21 AA32	LD	HL,IEXEC	
A95E	22 005E	LD	(005EH),HL	;SET VECTOR
A961	FB	EI		
;				
A962	E1	PRET: POP	HL	
A963	C9	RET		
;				
;STOP SOUND <-PSG MUTE <- "ctrl-D"				
A964	01 0707	QUIET1: LD	BC,CTC+3	
A967	3E 03	LD	A,3	
A969	ED 79	OUT	(C),A	;RESET CTC
;				
A96B	16 07	LD	D,7	
A96D	3E 08	LD	A,8	
A96F	CD ADD5	Q2: CALL	WOPM	
A972	15	DEC	D	
A973	20 FA	JR	NZ,Q2	
A975	CD ADD5	CALL	WOPM	;D=0
;				
A978	01 1C00	LD	BC,1C00H	
A97B	C3 013F	JP	013FH	;PATCH BACK
;				
;===== DO STRING =====				
A97E	CD 7FC3	DOSTR: CALL	7FC3H	;GET REAL ADDR.
A981	B7	OR	A	
A982	28 03	JR	Z,PRETX	;LEN=0
A984	CD A99F	CALL	STORE	;A=LEN
A987	E1	PRETX: POP	HL	
A988	7E	LD	A,(HL)	
A989	D6 3B	SUB	' ; '	;MUSIC "...";
A98B	23	INC	HL	
A98C	C8	RET	Z	
A98D	2B	DEC	HL	;BACK HL
A98E	ED 4B A9D6	PRETX: LD	BC,(HEAD0)	
A992	AF	XOR	A	
A993	ED 79	OUT	(C),A	;SET END MARK
A995	03	INC	BC	
A996	ED 43 A9D6	LD	(HEAD0),BC	
A99A	ED 43 A9D4	LD	(HEAD),BC	;COPY
A99E	C9	RET		
;				
A99F	ED 4B A9D6	STORE: LD	BC,(HEAD0)	
A9A3	2A A9D4	LD	HL,(HEAD)	
A9A6	B7	OR	A	
A9A7	ED 42	SBC	HL,BC	
A9A9	20 05	JR	NZ,STORE1	
A9AB	2E 3A	LD	L,' ; '	
A9AD	ED 69	OUT	(C),L	
A9AF	03	INC	BC	
A9B0	6F	STORE1: LD	L,A	;L=LEN
A9B1	1A	SLOOP: LD	A,(DE)	







AA68 ED 43 A9D2  
AA6C 21 AEA8  
AA6F E5  
AA70 21 AEB3  
AA73 16 00  
AA75 ED 78  
AA77 28 05  
AA79 03  
AA7A FE 3A  
AA7C 20 F7

AA7E B7  
AA7F 28 02  
AA81 3E 01  
AA83 E3  
AA84 77  
AA85 23  
AA86 E3  
AA87 71  
AA88 23  
AA89 70  
AA8A 23  
AA8B 14  
AA8C 7A  
AA8D FE 0B  
AA8F 38 E4  
AA91 E1

AA92 11 0000  
AA95 21 AEA8  
AA98 D5  
AA99 E5  
AA9A 7E  
AA9B B7  
AA9C C4 AAB3  
AA9F E1  
AAA0 D1  
AAA1 23  
AAA2 1C  
AAA3 7B  
AAA4 FE 0B  
AAA6 20 F0

AAA8 E1  
AAA9 D1  
AAAA C1  
AAAB F1  
AAAC ED 7B AA30  
AAB0 FB  
AAB1 ED 4D

AAB3 E5  
AAB4 D5  
AAB5 F2 AACA

AAB8 CD ADE0

AABB 78  
AABC B1  
AABD D1  
AABE E5  
AABF CC ACDB  
AAC2 E1  
AAC3 7C  
AAC4 B5  
AAC5 E1  
AAC6 C0  
AAC7 36 01  
AAC9 C9

AACA CB 23  
AACC 21 AEB3  
AACF 19  
AAD0 4E  
AAD1 23

```

;
START: LD (TAIL),BC
      LD HL,ACTF
      PUSH HL
      LD HL,PC
      LD D,0
SPC0: IN A,(C)
      JR Z,SPC1 ;HIT STRING END
      INC BC
      CP ':'
      JR NZ,SPC0
;
SPC1: OR A
      JR Z,SPC2
      LD A,1
SPC2: EX (SP),HL
      LD (HL),A ;!!
      INC HL ;INC ACTF ADDR.
      EX (SP),HL
      LD (HL),C
      INC HL
      LD (HL),B
      INC HL
      INC D
      LD A,D
      CP 11 ;CHECK COUNTER
      JR C,SPC0
      POP HL ;DROP ACTF
;
;PCS ARE READY
;00H=DEAD,01H=ALIVE,80H=ACTIVE
IE1: LD DE,0 ;CHANNEL
      LD HL,ACTF
IEL: PUSH DE
      PUSH HL
      LD A,(HL)
      OR A
      CALL NZ,ACT ;ALIVE OR ACTIVE
      POP HL
      POP DE
      INC HL
      INC E
      LD A,E
      CP 11
      JR NZ,IEL
;
IE9: POP HL
      POP DE
      POP BC
      POP AF
      LD SP,(INTSP)
      EI
      RETI ;RET FROM INT
;
;===== SUB =====
;
ACT: PUSH HL ;SAVE ACTF
      PUSH DE ;SAVE CH NO.
      JP P,ACT7
;
      CALL RCTR ;READ COUNTER
;BC=OMOTE,HL=URA
      LD A,B
      OR C
      POP DE
      PUSH HL
      CALL Z,KEYOFF
      POP HL
      LD A,H
      OR L
      POP HL
      RET NZ
      LD (HL),1 ;SLEEP
      RET
;
;ALIVE AND NOT-ACTIVE
ACT7: SLA E ;DE=DE*2
      LD HL,PC
      ADD HL,DE
      LD C,(HL)
      INC HL

```



AAD2	46		LD	B, (HL)	;GET PC
AAD3	ED 78				
AAD5	28 04		IN	A, (C)	;READ NOTE
AAD7	FE 3A		JR	Z, DIE	
AAD9	20 08		CP	' :	
			JR	NZ, WAKE	;HIT NOTE
AADB	D1				
AADC	70		DIE: POP	DE	
AADD	2B		DIE2: LD	(HL), B	
AADE	71		DEC	HL	
			LD	(HL), C	;SAVE PC
AADF	E1		POP	HL	;GET ACTF
AAE0	36 00		LD	(HL), 0	;KILL (ACTF)
AAE2	C9		RET		
			;WAKE UP		
AAE3	D1		WAKE: POP	DE	;DE=CHANNEL NO.
AAE4	ED 78		WUL: IN	A, (C)	
AAE6	28 F4		JR	Z, DIE2	
AAE8	FE 3A		CP	' :	
AAEA	28 F0		JR	Z, DIE2	
AAEC	D5		PUSH	DE	
AAED	E5		PUSH	HL	;SAVE PC.ADDR.
AAEE	CD AAFE		CALL	DONOTE	;DO NOTE
AAF1	E1		POP	HL	
AAF2	28 E7		JR	Z, DIE	;HIT ' :' OR 00H
AAF4	D1		POP	DE	;GET CH NO.
AAF5	38 ED		JR	C, WUL	;NOT REAL NOTE
			;HIT REAL NOTE		;A-G, N, W
AAF7	70		LD	(HL), B	
AAF8	2B		DEC	HL	
AAF9	71		LD	(HL), C	;SAVE PC
AAFA	E1		POP	HL	;GET ACTF.ADDR
AAFB	36 80		LD	(HL), 80H	;MAKE ACTIVE
AAFD	C9		RET		
			===== DO NOTE =====		
			;DE=CHANNEL NO. (BREAK OK)		
AAFE	ED 78		DONOTE: IN	A, (C)	
AB00	03		INC	BC	
AB01	FE 41		CP	' A'	
AB03	38 2A		JR	C, DN0	;?<A
AB05	FE 48		CP	' G'+1	
AB07	30 34		JR	NC, DN1	;?>G
AB09	CD AB1A		CALL	GETNT	;GET A-G
AB0C	C5		RN0: PUSH	BC	
AB0D	D5		PUSH	DE	
AB0E	CD AC67		CALL	KEYON	
AB11	D1		POP	DE	
AB12	C1		POP	BC	
AB13	CD ACED		RN1: CALL	DONUM	;NEW BC
AB16	3E 01		RN2: LD	A, 1	
AB18	B7		OR	A	;NZ, NC
AB19	C9		RET		
			;GET A-G		
AB1A	D6 41		GETNT: SUB	' A'	
AB1C	87		ADD	A, A	
AB1D	6F		LD	L, A	;SAVE NOTE
AB1E	ED 78		IN	A, (C)	
AB20	03		INC	BC	
AB21	FE 2D		CP	' -'	
AB23	C8		RET	Z	
AB24	2C		INC	L	
AB25	2C		INC	L	
AB26	FE 2B		CP	' +'	
AB28	C8		RET	Z	
AB29	FE 23		CP	' #'	
AB2B	C8		RET	Z	
AB2C	2D		DEC	L	;NOMAL NOTE
AB2D	0B		DEC	BC	
AB2E	C9		RET		
			;L=CODE,		
			;===== &, ( )		
			;=====		



AB2F	FE 3E	; &, >, <, DN0:	CP	'>'	; OCT UP
AB31	CA AC21		JP	Z, UPDOWN	
AB34	FE 3C		CP	'<'	; OCT DOWN
AB36	CA AC21		JP	Z, UPDOWN	
AB39	7F				
AB3A	B7	ACT8:	LD	A, A	
AB3B	37		OR	A	
AB3C	C9		SCF		
			RET		; NZ, C
AB3D	FE 4E	; R, N, Q, L, T, O, V, I, Y, W			
AB3F	28 51	DN1:	CP	'N'	; NOTE BY CODE
			JR	Z, NPLAY	
AB41	FE 52				
AB43	28 CE		CP	'R'	
			JR	Z, RN1	; Rn
AB45	FE 49				
AB47	CA AEF5		CP	'I'	; GAKKI BANGOU
			JP	Z, INST0	
AB4A	FE 59				
AB4C	CA ABF3		CP	'Y'	; 0-7:OPM, 8-10:PSG
			JP	Z, WREG	
AB4F	FE 57				
AB51	CA ABDD		CP	'W'	; BIG OMOTE, SMALL URA
			JP	Z, WAIT	
AB54	FE 54				
AB56	CA AC33		CP	'T'	; TEMPO
			JP	Z, STRUN	; RUNTIME SET TEMPO
AB59	21 0000				
AB5C	FE 4F		LD	HL, 0	
AB5E	28 0F		CP	'O'	
AB60	2C		JR	Z, SETGO	
AB61	FE 56		INC	L	
AB63	28 0A		CP	'V'	
AB65	2C		JR	Z, SETGO	
AB66	FE 51		INC	L	
AB68	28 05		CP	'Q'	; RATIO
AB6A	2C		JR	Z, SETGO	
AB6B	FE 4C		INC	L	
AB6D	20 CA		CP	'L'	; DEFAULT LEN
			JR	NZ, ACT8	; NEXT
AB6F	E5				
AB70	CD AE0E	SETGO:	PUSH	HL	
AB73	D1		CALL	DFLTBL	
AB74	19		POP	DE	
AB75	E5		ADD	HL, DE	
AB76	F5		PUSH	HL	
AB77	CD AD90		PUSH	AF	
AB7A	30 04		CALL	NUMBER	
AB7C	F1		JR	NC, STG1	
AB7D	D1		POP	AF	
AB7E	18 B9		POP	DE	
			JR	ACT8	
AB80	67				
AB81	F1	STG1:	LD	H, A	; SAVE LAST A
AB82	D1		POP	AF	
AB83	EB		POP	DE	
AB84	73		EX	DE, HL	
AB85	FE 4C		LD	(HL), E	
AB87	20 B0		CP	'L'	
AB89	7A		JR	NZ, ACT8	; RET WITH C, NZ
AB8A	FE 02		LD	A, D	
AB8C	20 AB		CP	'0'-'.'	
AB8E	CB FE		JR	NZ, ACT8	
AB90	18 A7		SET	7, (HL)	; SET BIT7
			JR	ACT8	
AB92	D5				
AB93	CD AD90	NPLAY:	PUSH	DE	
AB96	38 3F		CALL	NUMBER	
AB98	7D		JR	C, NPLAY9	
AB99	FE 60		LD	A, L	
AB9B	30 3A		CP	96	
			JR	NC, NPLAY9	
AB9D	1E 00				
AB9F	D6 0C		; A=A MOD 12, E=A/12		
ABA1	38 03	NPL:	LD	E, 0	
ABA3	1C		SUB	12	
ABA4	18 F9		JR	C, NPLAY0	
ABA6	C6 11		INC	E	
			JR	NPL	
		NPLAY0:	ADD	A, 12+5	



ABA8	FE 0A	CP	10
ABAA	38 07	JR	C,NPLAY1
ABAC	3C	INC	A
ABAD	FE 0F	CP	15
ABAF	38 02	JR	C,NPLAY1
ABB1	D6 0E	SUB	14
ABB3	57	NPLAY1: LD	D,A
ABB4	1C	INC	E ;D=CODE,E=OCT
ABB5	EB	EX	DE,HL
ABB6	D1	POP	DE
ABB7	D5	PUSH	DE ;PUSH CH
ABB8	E5	PUSH	HL ;PUSH CODE,OCT
ABB9	CD AE0E	CALL	DFLTBL
ABBC	22 ABDB	LD	(NPLAYW),HL
ABBF	D1	POP	DE
ABC0	7E	LD	A,(HL) ;PICK OCT
ABC1	73	LD	(HL),E ;SET OCT
		;HL=CH NO.	
ABC2	6A	LD	L,D
ABC3	D1	POP	DE
ABC4	F5	PUSH	AF
ABC5	C5	PUSH	BC
ABC6	D5	PUSH	DE
ABC7	CD AC67	CALL	KEYON
ABCA	D1	POP	DE
ABCB	C1	POP	BC
ABCC	CD ACED	CALL	DONUM
ABCF	F1	POP	AF
ABD0	2A ABDB	LD	HL,(NPLAYW)
ABD3	77	LD	(HL),A ;BACK OCT
ABD4	C3 AB16	JP	RN2
		;NPLAY9:	
ABD7	D1	POP	DE
ABD8	C3 AB39	JP	ACT8
		;NPLAYW:	
ABDB		DS	2
		;WAIT:	
ABDD	D5	PUSH	DE
ABDE	CD ACED	CALL	DONUM
ABE1	D1	POP	DE
ABE2	D5	PUSH	DE
ABE3	C5	PUSH	BC
ABE4	CD ADE0	CALL	RCTR
ABE7	23	INC	HL
ABE8	01 FFFF	LD	BC,0FFFFH
ABEB	CD ADFC	CALL	WCTR
ABEE	C1	POP	BC
ABEF	D1	POP	DE
ABF0	C3 AB16	JP	RN2
		;WREG:	
ABF3	D5	PUSH	DE
ABF4	CD AD90	CALL	NUMBER
ABF7	38 24	JR	C,WREG9
ABF9	ED 78	IN	A,(C)
ABFB	FE 2C	CP	' , '
ABFD	20 1E	JR	NZ,WREG9
ABFF	03	INC	BC
AC00	E5	PUSH	HL ;SAVE REG NO.
AC01	CD AD90	CALL	NUMBER
AC04	7D	LD	A,L
AC05	E1	POP	HL
AC06	38 15	JR	C,WREG9
AC08	D1	POP	DE
		;L=REG NO.,A=DATA,E=CHANNEL	
AC09	C5	PUSH	BC
AC0A	57	LD	D,A ;DATA
AC0B	7B	LD	A,E
AC0C	FE 08	CP	8
AC0E	7D	LD	A,L
AC0F	38 07	JR	C,WROP
AC11	CD ADC3	CALL	WPSG
AC14	C1	WREG1: POP	BC
AC15	C3 AB39	JP	ACT8
		;WRITE REGISTER (OPM)	
AC18	CD ADD5	WROP: CALL	WOPM
AC1B	18 F7	JR	WREG1
		;WREG9:	
AC1D	D1	POP	DE
AC1E	C3 AB39	JP	ACT8
		;	



AC21	D6 3D		UPDOWN:	SUB	'<'+1
AC23	CD AE0E			CALL	DFLTBL
AC26	86			ADD	A,(HL)
AC27	CA AB39			JP	Z,ACT8
AC2A	FE 09			CP	9
AC2C	D2 AB39			JP	NC,ACT8
AC2F	77			LD	(HL),A
AC30	C3 AB39			JP	ACT8
			:		
			;TEMPO		
AC33	CD AD90		STRUN:	CALL	NUMBER
AC36	DA AB39			JP	C,ACT8
AC39	7D			LD	A,L
AC3A	FE 1E			CP	30
AC3C	DA AB39			JP	C,ACT8 ;TOO SMALL
AC3F	C5			PUSH	BC
			:		
AC40	4D			LD	C,L
AC41	06 00			LD	B,0
AC43	11 0E4E			LD	DE,3662
AC46	21 0000			LD	HL,0
AC49	3E 10			LD	A,16
AC4B	CB 23		STR1:	SLA	E
AC4D	CB 12			RL	D
AC4F	ED 6A			ADC	HL,HL
AC51	B7			OR	A
AC52	ED 42			SBC	HL,BC
AC54	30 02			JR	NC,STR2
AC56	09			ADD	HL,BC ;BACK
AC57	1D			DEC	E
AC58	1C		STR2:	INC	E
AC59	3D			DEC	A
AC5A	20 EF			JR	NZ,STR1
			:		
AC5C	7B			LD	A,E
AC5D	32 AE17			LD	(TMPV),A
AC60	CD A9E7			CALL	TOCTC1
			:		
AC63	C1			POP	BC
AC64	C3 AB39			JP	ACT8
			:		
			;L=CODE (A-,A,A+,.....G,G+):0-14		
AC67	7B		KEYON:	LD	A,E ;CHANNEL NO.
AC68	FE 08			CP	8
AC6A	38 3A			JR	C,OPMDO
			:		
AC6C	7D		PSGDO:	LD	A,L ;A=CODE
AC6D	D5			PUSH	DE
AC6E	CD AE0E			CALL	DFLTBL
AC71	46			LD	B,(HL) ;GET OCT
AC72	23			INC	HL
AC73	4E			LD	C,(HL) ;VOL
AC74	87			ADD	A,A
AC75	5F			LD	E,A ;D=0
AC76	21 AE18			LD	HL,PSGFRQ
AC79	19			ADD	HL,DE
AC7A	56			LD	D,(HL)
AC7B	23			INC	HL
AC7C	5E			LD	E,(HL) ;ED=FREQ
AC7D	CB 3B		PSG1:	SRL	E
AC7F	CB 1A			RR	D ;DE=DE/2
AC81	10 FA			DJNZ	PSG1
			:		
AC83	E1			POP	HL ;HL=CHANNEL NO.
AC84	61			LD	H,C ;SAVE VOL
			:		
			;FREQ		
AC85	7D			LD	A,L
AC86	D6 08			SUB	8 ;A=FREQ LOW
AC88	87			ADD	A,A
AC89	CD ADC3			CALL	WPSG
AC8C	3C			INC	A ;A=FREQ HIGH
AC8D	53			LD	D,E
AC8E	CD ADC3			CALL	WPSG ;SET FREQ
			:		
			;VOL		
AC91	7D			LD	A,L ;CH8 -> R8
AC92	54			LD	D,H ;D=VOL
AC93	CD ADC3			CALL	WPSG
			:		
			;TONE?		



AC96	3E 07	LD	A, 7	
AC98	16 F8	LD	D, 0F8H	
AC9A	CD ADC3	CALL	WPSG	; A, B, C=TONE
AC9D	3E 0D	LD	A, 13	
AC9F	CD ADCC	CALL	RPSG	
ACA2	CD ADC3	CALL	WPSG	
ACA5	C9	RET		
;				
ACA6	7D	OPMDO:	LD	A, L
ACA7	C6 17		ADD	A, 17H
ACA9	FE 1B		CP	1BH
ACAB	38 0B		JR	C, OPMDO1
ACAD	D6 10		SUB	10H
ACAF	FE 0F		CP	0FH
ACB1	38 05		JR	C, OPMDO1
ACB3	FE 12		CP	12H
ACB5	28 01		JR	Z, OPMDO1
ACB7	3C		INC	A
ACB8	D5	OPMDO1:	PUSH	DE ;SAVE CH NO.
ACB9	CD AF52		CALL	OPMV0
ACBC	F5		PUSH	AF ;PUSH CODE
ACBD	CD AE0E		CALL	DFLTBL
ACC0	7E		LD	A, (HL) ;GET OCT....
ACC1	3D		DEC	A
ACC2	87		ADD	A, A
ACC3	87		ADD	A, A
ACC4	87		ADD	A, A
ACC5	87		ADD	A, A
ACC6	57		LD	D, A
ACC7	F1		POP	AF
ACC8	82		ADD	A, D ;GET CODE
ACC9	57		LD	D, A
ACCA	E1		POP	HL
ACCB	7D		LD	A, L ;A=CH NO.
ACCC	C6 28		ADD	A, 28H
ACCE	CD ADD5		CALL	WOPM
ACD1	7D		LD	A, L ;A=CH NO.
ACD2	F6 78		OR	78H
ACD4	57		LD	D, A
ACD5	3E 08		LD	A, 8
ACD7	CD ADD5		CALL	WOPM
ACDA	C9		RET	
;				
;E=CHANNEL NO.				
ACDB	7B	KEYOFF:	LD	A, E
ACDC	FE 08		CP	8
ACDE	38 06		JR	C, OFFOPM ;OPM
;				
ACE0	16 00		LD	D, 0
ACE2	CD ADC3		CALL	WPSG ;VOL=0
ACE5	C9		RET	
;				
ACE6	53	OFFOPM:	LD	D, E ;CH NO.
ACE7	3E 08		LD	A, 8
ACE9	CD ADD5		CALL	WOPM
ACEC	C9		RET	
;				
;E=CHANNEL NO.				
ACED	D5	DONUM:	PUSH	DE
ACEE	CD AD04		CALL	DONUMS
;RETURN HL=URA, DE=OMOTE, BC=NEW				
ACF1	2B		DEC	HL
ACF2	1B		DEC	DE
ACF3	ED 53 AD02		LD	(DONUMW), DE
ACF7	D1		POP	DE
ACF8	C5		PUSH	BC
ACF9	ED 4B AD02		LD	BC, (DONUMW) ;BC=OMOTE
ACFD	CD ADFC		CALL	WCTR
AD00	C1		POP	BC
AD01	C9		RET	
;				
AD02		DONUMW:	DS	2
;				
AD04	CD AE0E	DONUMS:	CALL	DFLTBL
AD07	23		INC	HL
AD08	23		INC	HL
AD09	56		LD	D, (HL)
AD0A	23		INC	HL
AD0B	5E		LD	E, (HL) ;D=Q, E=LEN
AD0C	D5		PUSH	DE
AD0D	ED 78		IN	A, (C)



AD0F	FE 40	CP	'@'
AD11	20 21	JR	NZ, DONOML
		;DIRECT	COUNTER SITEI
AD13	03	INC	BC
AD14	11 0000	LD	DE, 0
AD17	CD ADB0	XNUM0:	CALL DIGIT
AD1A	30 05	JR	NC, XNUMZ
AD1C	11 0001	LD	DE, 1
AD1F	18 41	JR	XXX
AD21	5F	XNUMZ:	LD E, A ;D=0
AD22	CD ADB0	XNUML:	CALL DIGIT
AD25	38 3B	JR	C, XXX
AD27	62	LD	H, D
AD28	6B	LD	L, E
AD29	29	ADD	HL, HL
AD2A	29	ADD	HL, HL
AD2B	19	ADD	HL, DE
AD2C	29	ADD	HL, HL
AD2D	5F	LD	E, A
AD2E	16 00	LD	D, 0
AD30	19	ADD	HL, DE
AD31	EB	EX	DE, HL
AD32	18 EE	JR	XNUML
AD34	CD AD90	DONOML:	CALL NUMBER ;
AD37	30 09	JR	NC, DONUM0
		;USE DEFAULT VALUE	
AD39	6B	LD	L, E
AD3A	CB 7D	BIT	7, L
AD3C	28 04	JR	Z, DONUM0
		;'. '=FUTEN !	
AD3E	CB BD	RES	7, L ;BIT7=0
AD40	3E FE	LD	A, 0FEH
AD42	F5	DONUM0:	PUSH AF
AD43	7D	LD	A, L
AD44	FE 21	CP	33
AD46	38 02	JR	C, DNOK
AD48	2E 20	LD	L, 32 ;MAX 32
AD4A	F1	DNOK:	POP AF
AD4B	26 00	LD	H, 0
AD4D	CB 25	SLA	L ;HL=HL*2
AD4F	11 AE36	LD	DE, LENTBL
AD52	19	ADD	HL, DE
AD53	5E	LD	E, (HL)
AD54	23	INC	HL
AD55	56	LD	D, (HL) ;DE=REAL LEN
AD56	FE FE	CP	0FEH
AD58	20 08	JR	NZ, DONUM1
AD5A	62	LD	H, D
AD5B	6B	LD	L, E
AD5C	CB 3A	SRL	D
AD5E	CB 1B	RR	E ;DE=DE/2
AD60	19	ADD	HL, DE ;HL=HL*1.5
AD61	EB	EX	DE, HL
AD62	62	XXX:	
AD62	62	DONUM1:	LD H, D ;DE=URA CTR
AD63	6B	LD	L, E ;COPY
AD64	F1	POP	AF ;A<-D=Q
		;KKKK	
AD65	E5	PUSH	HL
AD66	67	LD	H, A
AD67	ED 78	IN	A, (C)
AD69	FE 26	CP	'&'
AD6B	7C	LD	A, H
AD6C	E1	POP	HL
AD6D	20 0D	JR	NZ, DONUM2
AD6F	03	INC	BC
AD70	ED 78	IN	A, (C)
AD72	FE 2B	CP	'+'
AD74	20 18	JR	NZ, DONUM4
AD76	03	INC	BC
AD77	21 FFFF	LD	HL, 0FFFFH
AD7A	18 12	JR	DONUM4
AD7C	3D	DONUM2:	DEC A
AD7D	28 03	JR	Z, DONUM3
AD7F	19	ADD	HL, DE
AD80	18 FA	JR	DONUM2



AD82	CB 3C	;	DONUM3:	SRL	H	
AD84	CB 1D			RR	L	
AD86	CB 3C			SRL	H	
AD88	CB 1D			RR	L	
AD8A	CB 3C			SRL	H	
AD8C	CB 1D			RR	L	;HL=HL/8
			;			
			;	HL=OMOTE->BC,DE=URA->HL		
AD8E	EB		DONUM4:	EX	DE,HL	
AD8F	C9			RET		
			;	RETURN HL=URA,DE=OMOTE		
			;			
			;	L=VALUE,Carry=NOTHING,BC=AUTO INC		
			;	LAST='.' THEN Acc=FEH		
AD90	ED 78		NUMBER:	IN	A,(C)	
AD92	28 1A			JR	Z,NUM9	;HIT END
AD94	FE 3A			CP	'.'	
AD96	28 16			JR	Z,NUM9	;HIT END
			;			
AD98	CD ADB0		NUM0:	CALL	DIGIT	
AD9B	D8			RET	C	;ERR
			;	IKINARI '.' MO ERROR		
AD9C	6F			LD	L,A	
AD9D	CD ADB0		NUM1:	CALL	DIGIT	
ADA0	38 0A			JR	C,NUM1	
ADA2	67			LD	H,A	;SAVE
ADA3	7D			LD	A,L	
ADA4	87			ADD	A,A	;2
ADA5	87			ADD	A,A	;4
ADA6	85			ADD	A,L	;5
ADA7	87			ADD	A,A	;10
ADA8	84			ADD	A,H	
ADA9	6F			LD	L,A	
ADAA	18 F1			JR	NUM1	
			;			
ADAC	B7		NUM1:	OR	A	;NC
ADAD	C9			RET		
			;			
ADAE	37		NUM9:	SCF		
ADAF	C9			RET		
			;			
ADB0	ED 78		DIGIT:	IN	A,(C)	
ADB2	FE 3A			CP	'9'+1	
ADB4	38 02			JR	C,DIGIT1	
ADB6	37			SCF		
ADB7	C9			RET		
ADB8	D6 30		DIGIT1:	SUB	'0'	
ADBA	03			INC	BC	
ADBB	D0			RET	NC	
ADBC	FE FE			CP	'.'-'0'	
ADBE	28 01			JR	Z,DIGIT2	
ADC0	0B			DEC	BC	
ADC1	37		DIGIT2:	SCF		
ADC2	C9			RET		
			;			
ADC3	01 1C00		WPSG:	LD	BC,1C00H	
ADC6	ED 79			OUT	(C),A	
ADC8	05			DEC	B	
ADC9	ED 51			OUT	(C),D	
ADCB	C9			RET		
			;			
ADCC	01 1C00		RPSG:	LD	BC,1C00H	
ADCF	ED 79			OUT	(C),A	
ADD1	05			DEC	B	
ADD2	ED 50			IN	D,(C)	
ADD4	C9			RET		
			;			
ADD5	C5		WOPM:	PUSH	BC	
ADD6	01 0700			LD	BC,0700H	
ADD9	ED 79			OUT	(C),A	
ADDB	0C			INC	C	
ADDC	ED 51			OUT	(C),D	
ADDE	C1			POP	BC	
ADDF	C9			RET		
			;			
ADE0	D5		RCTR:	PUSH	DE	
ADE1	CB 23			SLA	E	
ADE3	CB 23			SLA	E	
ADE5	21 AEC9			LD	HL,PACK	
ADE8	19			ADD	HL,DE	



ADE9	4E	LD	C, (HL)
ADEA	23	INC	HL
ADEB	46	LD	B, (HL) ;BC=OMOTE
ADEC	23	INC	HL
ADED	5E	LD	E, (HL)
ADEE	23	INC	HL
ADEF	56	LD	D, (HL) ;DE=URA
ADF0	EB	EX	DE, HL ;HL=URA
ADF1	0B	DEC	BC
ADF2	2B	DEC	HL
ADF3	D1	POP	DE
ADF4	C5	PUSH	BC
ADF5	E5	PUSH	HL
ADF6	CD ADFC	CALL	WCTR
ADF9	E1	POP	HL
ADFA	C1	POP	BC
ADFB	C9	RET	
;			
ADFC	CB 23	WCTR:	SLA E
ADFE	CB 23		SLA E
AE00	E5		PUSH HL
AE01	21 AEC9		LD HL, PACK
AE04	19		ADD HL, DE
AE05	71		LD (HL), C
AE06	23		INC HL
AE07	70		LD (HL), B ;BC=OMOTE
AE08	23		INC HL
AE09	C1		POP BC ;BC=URA
AE0A	71		LD (HL), C
AE0B	23		INC HL
AE0C	70		LD (HL), B
AE0D	C9		RET
;			
AE0E	21 AE7C	DFLTBL:	LD HL, DFLW
AE11	CB 23		SLA E
AE13	CB 23		SLA E
AE15	19		ADD HL, DE
AE16	C9		RET
;			
;===== VAR AREA =====			
;			
AE17		TMPV:	DS 1
;			
AE18	12D0	PSGFRQ:	DW 12D0H ;Ab=G#
AE1A	11C2		DW 11C2H ;A
AE1C	10C4		DW 10C4H ;A#
AE1E	0FD2		DW 0FD2H ;B
AE20	0EEE		DW 0EEEH ;B#=(C)
AE22	1DDE		DW 1DDEH ;C
AE24	1C2E		DW 1C2EH ;C#
AE26	1A9A		DW 1A9AH ;D
AE28	191C		DW 191CH ;D#
AE2A	17B4		DW 17B4H ;E
AE2C	165E		DW 165EH ;E#=F
AE2E	165E		DW 165EH ;F
AE30	151E		DW 151EH ;F#
AE32	13EE		DW 13EEH ;G
AE34	12D0		DW 12D0H ;G#
;			
AE36	0800 0400	LENTBL:	DW 2048, 1024, 512, 341, 256
AE3A	0200 0155		
AE3E	0100		
AE40	00CC 00AA		DW 204, 170, 146, 128, 113
AE44	0092 0080		
AE48	0071		
AE4A	0066 005D		DW 102, 93, 85, 78, 73
AE4E	0055 004E		
AE52	0049		
AE54	0044 0040		DW 68, 64, 60, 56, 53
AE58	003C 0038		
AE5C	0035		
AE5E	0033 0030		DW 51, 48, 46, 44, 42
AE62	002E 002C		
AE66	002A		
AE68	0028 0027		DW 40, 39, 37, 36, 35
AE6C	0025 0024		
AE70	0023		
AE72	0022 0021		DW 34, 33, 32
AE76	0020		
;			
;OCT, VOL, Q, LEN: : 4, 100(12), 8, 4			



AE78 04 64 08 04  
 AE7C 04 64 08 04  
 AE80 04 64 08 04  
 AE84 04 64 08 04  
 AE88 04 64 08 04  
 AE8C 04 64 08 04  
 AE90 04 64 08 04  
 AE94 04 64 08 04  
 AE98 04 64 08 04  
 AE9C 04 0C 08 04  
 AEA0 04 0C 08 04  
 AEA4 04 0C 08 04

AEA8  
 AEB3  
 AEC9

AEF5 7B  
 AEF6 FE 08  
 AEF8 D2 AB39  
 AEFB D5  
 AEFC CD AD90  
 AEFF 38 48  
 AF01 2D  
 AF02 7D  
 AF03 FE 28  
 AF05 30 42

AF07 D1  
 AF08 D5  
 AF09 21 AFEC  
 AF0C 19  
 AF0D 77

AF0E 6F  
 AF0F CD AFC8

AF12 D1  
 AF13 C5  
 AF14 E5  
 AF15 3E 20  
 AF17 83  
 AF18 CD AF4D  
 AF1B C6 10  
 AF1D 16 14  
 AF1F CD ADD5  
 AF22 C6 08  
 AF24 CD AF4D

AF27 1E 06  
 AF29 CD AFD7  
 AF2C 1D  
 AF2D 20 FA

AF2F 11 0005  
 AF32 19  
 AF33 3E 18  
 AF35 CD AF4D  
 AF38 3C  
 AF39 CD AF4D  
 AF3C CD AF4D  
 AF3F 3E 1B  
 AF41 CD AF4D  
 AF44 E1  
 AF45 C1  
 AF46 C3 AB39

AF49 D1  
 AF4A C3 AB39

AF4D 56  
 AF4E 23  
 AF4F C3 ADD5

AF52 F5  
 AF53 D5

DFLDY: DB 4,100,8,4  
 DFLW: DB 4,100,8,4 ;0  
 DB 4,100,8,4 ;1  
 DB 4,100,8,4 ;2  
 DB 4,100,8,4 ;3  
 DB 4,100,8,4 ;4  
 DB 4,100,8,4 ;5  
 DB 4,100,8,4 ;6  
 DB 4,100,8,4 ;7  
 DB 4,12,8,4 ;8  
 DB 4,12,8,4 ;9  
 DB 4,12,8,4 ;10

ACTF: DS 11  
 PC: DS 22  
 PACK: DS 44

;  
 ;.....  
 ;  
 ;DATA=B000H-B72FH (10\*40+36\*40)

INST0: LD A,E  
 CP 8  
 JP NC,ACT8 ;PSG  
 PUSH DE ;SAVE CH NO.  
 CALL NUMBER  
 JR C,INST9  
 DEC L ;L=L-1  
 LD A,L  
 CP 40  
 JR NC,INST9

;  
 POP DE  
 PUSH DE  
 LD HL,INSTN  
 ADD HL,DE  
 LD (HL),A ;SAVE INSTN

;  
 LD L,A  
 CALL GETVTD

;  
 ;20,38,(40,60,80,A0,C0,E0)\*4,(UD+KSL)\*4  
 ;KTRNS,18H,PMD,AMD,1BH,DLY+SYNC

POP DE  
 PUSH BC ;SAVE PC  
 PUSH HL ;SAVE ADDR.  
 LD A,20H  
 ADD A,E  
 CALL WOPMX ;R/L/FB/ALG  
 ADD A,10H  
 LD D,14H  
 CALL WOPM ;KF  
 ADD A,08H  
 CALL WOPMX ;PMS/AMS

;  
 INST1: LD E,6  
 CALL WOPMXX  
 DEC E  
 JR NZ,INST1

;  
 LD DE,5  
 ADD HL,DE ;HL=HL+5  
 LD A,18H  
 CALL WOPMX ;LFRQ  
 INC A ;A=19H  
 CALL WOPMX ;PMD  
 CALL WOPMX ;AMD  
 LD A,1BH  
 CALL WOPMX ;FRM  
 POP HL  
 POP BC  
 JP ACT8

;  
 INST9: POP DE ;DROP  
 JP ACT8

;  
 WOPMX: LD D,(HL)  
 INC HL  
 JP WOPM

;  
 OPMV0: PUSH AF ;A=CODE  
 PUSH DE ;DE=CH NO.



AF54	D5		PUSH	DE	
AF55	CD AE0E		CALL	DFTLTL	
AF58	23		INC	HL	
AF59	7E		LD	A, (HL)	;OVQL
AF5A	D1		POP	DE	
AF5B	F5		PUSH	AF	;SAVE V
AF5C	21 AFEC		LD	HL, INSTN	
AF5F	19		ADD	HL, DE	
AF60	6E		LD	L, (HL)	;L=In
;					
AF61	CD AFC8		CALL	GETVTD	;GET ADDR.
AF64	7E		LD	A, (HL)	
AF65	E6 07		AND	7	;GET ALG
AF67	0E 07		LD	C, 0111B	;BIT3=0
AF69	FE 04		CP	4	
AF6B	38 0B		JR	C, OPMV1	;ALG0-3
AF6D	CB 89		RES	1, C	;OP2=CRY
AF6F	28 07		JR	Z, OPMV1	
AF71	CB 91		RES	2, C	;OP3=CRY
AF73	FE 07		CP	7	
AF75	20 01		JR	NZ, OPMV1	
AF77	0D		DEC	C	;BIT0=0
;					
AF78	11 0006	OPMV1:	LD	DE, 6	;C=MASK
AF7B	19		ADD	HL, DE	;POINT V
AF7C	11 AFC4		LD	DE, V4	
AF7F	06 04		LD	B, 4	
AF81	7E	OPMV2:	LD	A, (HL)	
AF82	CB 09		RRC	C	
AF84	30 02		JR	NC, OPMV3	
AF86	F6 80		OR	80H	;MODULE MARK
AF88	12	OPMV3:	LD	(DE), A	
AF89	23		INC	HL	
AF8A	13		INC	DE	
AF8B	10 F4		DJNZ	OPMV2	
;					
AF8D	06 04		LD	B, 4	
AF8F	EB		EX	DE, HL	
AF90	2B		DEC	HL	;HL=V4+3
AF91	3E 7F		LD	A, 07FH	
AF93	BE	OPMV4:	CP	(HL)	
AF94	38 01		JR	C, OPMV5	
;A >= (HL)					
AF96	7E		LD	A, (HL)	
AF97	2B	OPMV5:	DEC	HL	
AF98	10 F9		DJNZ	OPMV4	
;A=MIN TL					
;ADD TO ALL CRY '127-V-A', MIN=0, MAX=7F					
AF9A	C1		POP	BC	;B=V
AF9B	ED 44		NEG		
AF9D	90		SUB	B	
AF9E	C6 7F		ADD	A, 127	
AFA0	4F		LD	C, A	
;					
AFA1	D1		POP	DE	
AFA2	D5		PUSH	DE	;E=CH NO.
AFA3	3E 58		LD	A, 60H-8	
AFA5	83		ADD	A, E	
AFA6	F5		PUSH	AF	;SAVE REG#-8
;					
AFA7	23		INC	HL	;HL=V4
AFA8	E5		PUSH	HL	;SAVE V4
AFA9	06 04		LD	B, 4	
AFAB	7E	OPMV6:	LD	A, (HL)	
AFAC	B7		OR	A	
AFAD	FA AFB6		JP	M, OPMV7	;MUDULE
AFB0	81		ADD	A, C	
AFB1	F2 AFB6		JP	P, OPMV7	;0-7FH
AFB4	3E 7F		LD	A, 7FH	;MAX=7FH
AFB6	E6 7F	OPMV7:	AND	7FH	
AFB8	77		LD	(HL), A	
AFB9	23		INC	HL	
AFBA	10 EF		DJNZ	OPMV6	
;					
AFBC	E1		POP	HL	;HL=V4
AFBD	F1		POP	AF	;A=REG#-8
AFBE	CD AFD7		CALL	WOPMXX	
;					
AFC1	D1		POP	DE	
AFC2	F1		POP	AF	
AFC3	C9		RET		



AFC4		;	V4:	DS	4	
AFC8	26 00	;	GETVTD:	LD	H,0	
AFCA	29			ADD	HL,HL	;2
AFCB	29			ADD	HL,HL	;4
AFCC	54			LD	D,H	
AFCD	5D			LD	E,L	;COPY
AFCE	29			ADD	HL,HL	;8
AFCF	29			ADD	HL,HL	;16
AFD0	29			ADD	HL,HL	;32
AFD1	19			ADD	HL,DE	;36
AFD2	11 B190			LD	DE,0B190H	
AFD5	19			ADD	HL,DE	;GET VTD!
AFD6	C9			RET		
AFD7	C6 08		;	WOPMXX:	ADD	A,08H
AFD9	CD AF4D			CALL	WOPMX	
AFDC	C6 10			ADD	A,10H	
AFDE	CD AF4D			CALL	WOPMX	
AFE1	D6 08			SUB	08H	
AFE3	CD AF4D			CALL	WOPMX	
AFE6	C6 10			ADD	A,10H	
AFE8	CD AF4D			CALL	WOPMX	
AFEB	C9			RET		
AFEC	00 00 00 00		;	INSTN:	DB	0,0,0,0
AFF0	01 01 01 01				DB	1,1,1,1
			;			
				END		

リスト 11-5 MML ダンプリスト

A8B0	21 00	A9 22	E3 2D	22 11	:	2F	
A8B8	2E 22	13 2E	C9 4E	41 54	:	3D	
A8C0	48 41	4E 49	45 4C	20 53	:	24	
A8C8	54 49	54 54	1A 1A	1A 1A	:	AD	
A8D0	1A 1A	1A 1A	1A 1A	1A 1A	:	D0	
A8D8	1A 1A	1A 1A	1A 1A	1A 1A	:	D0	
A8E0	1A 1A	1A 1A	1A 1A	1A 1A	:	D0	
A8E8	1A 1A	1A 1A	1A 1A	1A 1A	:	D0	
A8F0	1A 1A	1A 1A	1A 1A	1A 1A	:	D0	
A8F8	1A 1A	1A 1A	1A 1A	1A 1A	:	D0	
A900	CD D1	7F 3A	DB A5	E5 EB	:	A7	
A908	FE 03	28 72	CD 36	54 CD	:	BF	
A910	AF 5B	7C B5	20 4C	21 01	:	C9	
A918	40 22	D4 A9	22 D6	A9 2B	:	AB	
A920	22 D2	A9 44	4D AF	ED 79	:	43	
A928	3E 1E	32 17	AE 21	78 AE	:	9A	
-----							
SUM:	A1 89	CC EE	8C 4A	A1 79	59F4		
A930	11 7C	AE 01	24 00	ED B0	:	FD	
A938	23 36	0C 2B	01 08	00 ED	:	86	
A940	B0 EB	06 0B	36 00	23 10	:	15	
A948	FB F3	3E C3	32 3C	01 21	:	7F	
A950	64 A9	22 3D	01 CD	3C 01	:	77	
A958	CD D8	A9 21	32 AA	22 5E	:	CB	
A960	00 FB	E1 C9	01 07	07 3E	:	F2	
A968	03 ED	79 16	07 3E	08 CD	:	99	
A970	D5 AD	15 20	FA CD	D5 AD	:	00	
A978	01 00	1C C3	3F 01	CD C3	:	B0	
A980	7F B7	28 03	CD 9F	A9 E1	:	57	
A988	7E D6	3B 23	C8 2B	ED 4B	:	DD	
A990	D6 A9	AF ED	79 03	ED 43	:	C7	
A998	D6 A9	ED 43	D4 A9	C9 ED	:	E2	
A9A0	4B D6	A9 2A	D4 A9	B7 ED	:	15	
A9A8	42 20	05 2E	3A ED	69 03	:	28	
-----							
SUM:	1F 7B	01 C8	F1 DA	8C F4	7EDA		
A9B0	6F 1A	FE 61	38 06	FE 7B	:	9F	
A9B8	30 02	C6 E0	ED 79	03 13	:	54	
A9C0	78 B1	28 08	2D 20	EA ED	:	7D	
A9C8	43 D6	A9 C9	CD 3C	01 C3	:	58	
A9D0	27 20	00 40	00 40	00 40	:	07	
A9D8	01 04	07 3E	27 ED	79 3E	:	15	
A9E0	01 ED	79 3E	58 ED	79 01	:	64	
A9E8	07 07	3E C7	ED 79	3A 17	:	CA	
A9F0	AE ED	79 C9	0A 09	49 4E	:	87	
A9F8	43 09	44 45	0D 0A	09 44	:	39	
AA00	4A 4E	5A 09	4F 50	4D 56	:	3D	
AA08	32 CA	03 FF	03 00	00 00	:	01	
AA10	00 00	00 00	00 00	00 00	:	00	
AA18	00 00	00 00	00 00	00 00	:	00	
AA20	00 00	00 00	00 00	00 CA	:	CA	
AA28	56 D9	00 00	00 00	C3 BE	:	B0	
-----							
SUM:	4D A2	6D AB	F4 D1	7A 44	C55D		
AA30	39 04	F3 ED	73 30	AA 31	:	9B	
AA38	30 AA	F5 C5	D5 E5	21 A8	:	17	
AA40	AE 06	0B AF	B6 23	20 4A	:	B1	
AA48	10 FA	ED 4B	D2 A9	2A D4	:	BB	
AA50	A9 2B	B7 ED	42 28	51 ED	:	20	
AA58	78 03	28 0C	ED 78	03 20	:	37	
AA60	FB 0B	ED 43	D2 A9	18 E6	:	AF	
AA68	ED 43	D2 A9	21 A8	AE E5	:	07	
AA70	21 B3	AE 16	00 ED	78 28	:	25	
AA78	05 03	FE 3A	20 F7	B7 28	:	36	
AA80	02 3E	01 E3	77 23	E3 71	:	12	
AA88	23 70	23 14	7A FE	0B 38	:	85	
AA90	E4 E1	11 00	00 21	A8 AE	:	4D	
AA98	D5 E5	7E B7	C4 B3	AA E1	:	F1	
AAA0	D1 23	1C 7B	FE 0B	20 F0	:	A4	
AAA8	E1 D1	C1 F1	ED 7B	30 AA	:	A6	
-----							
SUM:	E6 48	BA FB	B2 31	EE F1	45E6		



```

AAB0 FB ED 4D E5 D5 F2 CA AA : 55
AAB8 CD E0 AD 78 B1 D1 E5 CC : 05
AAC0 DB AC E1 7C B5 E1 C0 36 : 70
AAC8 01 C9 CB 23 21 B3 AE 19 : 53
AAD0 4E 23 46 ED 78 28 04 FE : 46
AAD8 3A 20 08 D1 70 2B 71 E1 : 20
AAE0 36 00 C9 D1 ED 78 28 F4 : 51
AAE8 FE 3A 28 F0 D5 E5 CD FE : D5
AAF0 AA E1 28 E7 D1 38 ED 70 : 00
AAF8 2B 71 E1 36 80 C9 ED 78 : 61
AB00 03 FE 41 38 2A FE 48 30 : 1A
AB08 34 CD 1A AB C5 D5 CD 67 : 94
AB10 AC D1 C1 CD ED AC 3E 01 : E3
AB18 B7 C9 D6 41 87 6F ED 78 : F2
AB20 03 FE 2D C8 2C 2C FE 2B : 77
AB28 C8 FE 23 C8 2D 0B C9 FE : B0
-----
SUM: 9A 72 30 19 13 2D 68 B7 AC0B

AB30 3E CA 21 AC FE 3C CA 21 : FA
AB38 AC 7F B7 37 C9 FE 4E 28 : 56
AB40 51 FE 52 28 CE FE 49 CA : A8
AB48 F5 AE FE 59 CA F3 AB FE : 60
AB50 57 CA DD AB FE 54 CA 33 : F8
AB58 AC 21 00 00 FE 4F 28 0F : 51
AB60 2C FE 56 28 0A 2C FE 51 : 2D
AB68 28 05 2C FE 4C 20 CA E5 : 72
AB70 CD 0E AE D1 19 E5 F5 CD : 1A
AB78 90 AD 30 04 F1 D1 18 B9 : 04
AB80 67 F1 D1 EB 73 FE 4C 20 : F1
AB88 B0 7A FE 02 20 AB CB FE : BE
AB90 18 A7 D5 CD 90 AD 38 3F : 15
AB98 7D FE 60 30 3A 1E 00 D6 : 39
ABA0 0C 38 03 1C 18 F9 C6 11 : 4B
ABA8 FE 0A 38 07 3C FE 0F 38 : C8
-----
SUM: 9A F0 A4 17 6C 3B F7 8B 113D

ABB0 02 D6 0E 57 1C EB D1 D5 : EA
ABB8 E5 CD 0E AE 22 DB AB D1 : E7
ABC0 7E 73 6A D1 F5 C5 D5 CD : 88
ABC8 67 AC D1 C1 CD ED AC F1 : FC
ABD0 2A DB AB 77 C3 16 AB D1 : 7C
ABD8 C3 39 AB 0A 53 D5 CD ED : 93
ABE0 AC D1 D5 C5 CD E0 AD 23 : 94
ABE8 01 FF FF CD FC AD C1 D1 : 07
ABF0 C3 16 AB D5 CD 90 AD 38 : 9B
ABF8 24 ED 78 FE 2C 20 1E 03 : F4
AC00 E5 CD 90 AD 7D E1 38 15 : 9A
AC08 D1 C5 57 7B FE 08 7D 38 : 23
AC10 07 CD C3 AD C1 C3 39 AB : AC
AC18 CD D5 AD 18 F7 D1 C3 39 : 2B
AC20 AB D6 3D CD 0E AE 86 CA : 97
AC28 39 AB FE 09 D2 39 AB 77 : 18
-----
SUM: BB 5E 36 40 EB 04 90 C3 540F

AC30 C3 39 AB CD 90 AD DA 39 : C4
AC38 AB 7D FE 1E DA 39 AB C5 : C7
AC40 4D 06 00 11 4E 0E 21 00 : E1
AC48 00 3E 10 CB 23 CB 12 ED : 06
AC50 6A B7 ED 42 30 02 09 1D : A8
AC58 1C 3D 20 EF 7B 32 17 AE : DA
AC60 CD E7 A9 C1 C3 39 AB 7B : 40
AC68 FE 08 38 3A 7D D5 CD 0E : A5
AC70 AE 46 23 4E 87 5F 21 18 : 84
AC78 AE 19 56 23 5E CB 3B CB : 6F
AC80 1A 10 FA E1 61 7D D6 08 : C1
AC88 87 CD C3 AD 3C 53 CD C3 : E3

```

```

AC90 AD 7D 54 CD C3 AD 3E 07 : 00
AC98 16 F8 CD C3 AD 3E 0D CD : 63
ACA0 CC AD CD C3 AD C9 7D C6 : C2
ACA8 17 FE 1B 38 0B D6 10 FE : 57
-----
SUM: AF 39 E6 7D 70 85 27 85 4F0B

ACB0 0F 38 05 FE 12 28 01 3C : C1
ACB8 D5 CD 52 AF F5 CD 0E AE : 21
ACC0 7E 3D 87 87 87 87 57 F1 : 1F
ACC8 82 57 E1 7D C6 28 CD D5 : C7
ACD0 AD 7D F6 78 57 3E 08 CD : 02
ACD8 D5 AD C9 7B FE 08 38 06 : 0A
ACE0 16 00 CD C3 AD C9 53 3E : AD
ACE8 08 CD D5 AD C9 D5 CD 04 : C6
ACF0 AD 2B 1B ED 53 02 AD D1 : B3
ACF8 C5 ED 4B 02 AD CD FC AD : 22
AD00 C1 C9 4F 20 CD 0E AE 23 : A5
AD08 23 56 23 5E D5 ED 78 FE : 32
AD10 40 20 21 03 11 00 00 CD : 62
AD18 B0 AD 30 05 11 01 00 18 : BC
AD20 41 5F CD B0 AD 38 3B 62 : 9F
AD28 6B 29 29 19 29 5F 16 00 : 74
-----
SUM: 76 1C 3F 52 B9 EA B3 AB 3973

AD30 19 EB 18 EE CD 90 AD 30 : 44
AD38 09 6B CB 7D 28 04 CB BD : 70
AD40 3E FE F5 7D FE 21 38 02 : 07
AD48 2E 20 F1 26 00 CB 25 11 : 66
AD50 36 AE 19 5E 23 56 FE FE : D0
AD58 20 08 62 6B CB 3A CB 1B : E0
AD60 19 EB 62 6B F1 E5 67 ED : FB
AD68 78 FE 26 7C E1 20 0D 03 : 29
AD70 ED 78 FE 2B 20 18 03 21 : EA
AD78 FF FF 18 12 3D 28 03 19 : A9
AD80 18 FA CB 3C CB 1D CB 3C : 08
AD88 CB 1D CB 3C CB 1D EB C9 : 8B
AD90 ED 78 28 1A FE 3A 28 16 : 1D
AD98 CD B0 AD D8 6F CD B0 AD : 9B
ADA0 38 0A 67 7D 87 87 85 87 : 40
ADA8 84 6F 18 F1 B7 C9 37 C9 : 7C
-----
SUM: BA 42 CC D3 51 E6 62 5B A267

ADB0 ED 78 FE 3A 38 02 37 C9 : D7
ADB8 D6 30 03 D0 FE FE 28 01 : FE
ADC0 0B 37 C9 01 00 1C ED 79 : 8E
ADC8 05 ED 51 C9 01 00 1C ED : 16
ADD0 79 05 ED 50 C9 C5 01 00 : 4A
ADD8 07 ED 79 0C ED 51 C1 C9 : 41
ADE0 D5 CB 23 CB 23 21 C9 AE : 49
ADE8 19 4E 23 46 23 5E 23 56 : CA
ADF0 EB 0B 2B D1 C5 E5 CD FC : 65
ADF8 AD E1 C1 C9 CB 23 CB 23 : F4
AE00 E5 21 C9 AE 19 71 23 70 : 9A
AE08 23 C1 71 23 70 C9 21 7C : 4E
AE10 AE CB 23 CB 23 19 C9 33 : 9F
AE18 D0 12 C2 11 C4 10 D2 0F : 6A
AE20 EE 0E DE 1D 2E 1C 9A 1A : F5
AE28 1C 19 B4 17 5E 16 5E 16 : E8
-----
SUM: 69 A9 64 BC BF 4E 85 7A 0F37

AE30 1E 15 EE 13 D0 12 00 08 : 1E
AE38 00 04 00 02 55 01 00 01 : 5D
AE40 CC 00 AA 00 92 00 80 00 : 88
AE48 71 00 66 00 5D 00 55 00 : 89
AE50 4E 00 49 00 44 00 40 00 : 1B

```



```

AE58 3C 00 38 00 35 00 33 00 : DC
AE60 30 00 2E 00 2C 00 2A 00 : B4
AE68 28 00 27 00 25 00 24 00 : 98
AE70 23 00 22 00 21 00 20 00 : 86
AE78 04 64 08 04 04 64 08 04 : E8
AE80 04 64 08 04 04 64 08 04 : E8
AE88 04 64 08 04 04 64 08 04 : E8
AE90 04 64 08 04 04 64 08 04 : E8
AE98 04 64 08 04 04 0C 08 04 : 90
AEA0 04 0C 08 04 04 0C 08 04 : 38
AEA8 09 4C 44 09 42 43 2C 43 : 96

```

```

-----
SUM: 81 65 6A 36 59 FE 12 64 A752

```

```

AEB0 54 43 2B 33 0D 0A 09 4C : 61
AEB8 44 09 41 2C 33 0D 0A 09 : 0D
AEC0 4F 55 54 09 28 43 29 2C : C1
AEC8 41 09 3B 52 45 53 45 54 : 08
AED0 20 43 54 43 0D 0A 3B 0D : 59
AED8 0A 09 4C 44 09 44 2C 37 : 53
AEE0 0D 0A 09 4C 44 09 41 2C : 26
AEE8 38 0D 0A 51 32 3A 09 43 : 58
AEF0 41 4C 4C 09 57 7B FE 08 : BA
AEF8 D2 39 AB D5 CD 90 AD 38 : CD
AF00 48 2D 7D FE 28 30 42 D1 : 5B
AF08 D5 21 EC AF 19 77 6F CD : 5D
AF10 C8 AF D1 C5 E5 3E 20 83 : D3
AF18 CD 4D AF C6 10 16 14 CD : 96
AF20 D5 AD C6 08 CD 4D AF 1E : 37
AF28 06 CD D7 AF 1D 20 FA 11 : A1

```

```

-----
SUM: 37 56 2B AB 7D B1 6B E5 2AF1

```

```

AF30 05 00 19 3E 18 CD 4D AF : 3D
AF38 3C CD 4D AF CD 4D AF 3E : 0C
AF40 1B CD 4D AF E1 C1 C3 39 : 82
AF48 AB D1 C3 39 AB 56 23 C3 : 5F
AF50 D5 AD F5 D5 D5 CD 0E AE : AA
AF58 23 7E D1 F5 21 EC AF 19 : 3C
AF60 6E CD C8 AF 7E E6 07 0E : 2B
AF68 07 FE 04 38 0B CB 89 28 : C8
AF70 07 CB 91 FE 07 20 01 0D : 96
AF78 11 06 00 19 11 C4 AF 06 : BA
AF80 04 7E CB 09 30 02 F6 80 : FE
AF88 12 23 13 10 F4 06 04 EB : 41
AF90 2B 3E 7F BE 38 01 7E 2B : 88
AF98 10 F9 C1 ED 44 90 C6 7F : D0
AFA0 4F D1 D5 3E 58 83 F5 23 : 26
AFA8 E5 06 04 7E B7 FA B6 AF : 83

```

```

-----
SUM: 11 E1 90 1D B7 95 C8 E0 74FD

```

```

AFB0 81 F2 B6 AF 3E 7F E6 7F : FA
AFB8 77 23 10 EF E1 F1 CD D7 : 0F
AFC0 AF D1 F1 C9 0D 0A 50 52 : F3
AFC8 26 00 29 29 54 5D 29 29 : 7B
AFD0 29 19 11 90 B1 19 C9 C6 : 3C
AFD8 08 CD 4D AF C6 10 CD 4D : C1
AFE0 AF D6 08 CD 4D AF C6 10 : 2C
AFE8 CD 4D AF C9 00 00 00 00 : 92
AFF0 01 01 01 01 : 04

```

```

-----
SUM: 7B F0 F6 66 44 AF 88 F4 6E00

```

## リスト 11-6 サンプル曲(美しく青きドナウ)

```

100 TEMPO 0
110 MUSIC "I1 V127 Q4:I1 V127 Q4:I1 V127 Q4"
120 MUSIC "T150 O4 L4 C&E&G G2>G GRE E<RC C&E&G";
130 MUSIC ":O3 L4 RRR CGG CGG CGG CGG ";
140 MUSIC ":O3 L4 RRR REE REE REE REE ";
150 MUSIC " G2>G GRF FRO3B B&O4D&A A2>A";
160 MUSIC ":O2B>GG< B>GG< B>GG DGG< B>GG";
170 MUSIC ":O3RFF RFF RFF RFF RFF";
180 MUSIC " ARF FRO3B B&O4D&A A2>A ARE";
190 MUSIC ":O2B>GG< B>GG DGG CGG CGG";
200 MUSIC ":O3RFF RFF RFF REE REE";
210 MUSIC " ERO4C C&E&G >C2>C CR<G GRO4C C&E&G";
220 MUSIC ":O3CGG CGG CGG CGG CGG CGG";
230 MUSIC ":O3REE REE REE REE REE REE";
240 MUSIC " O5C2>C CR<A ARD D&F&A A1 F#G";
250 MUSIC ":O3DAA DAA ARR D&F&A <B>GG< B>GG";
260 MUSIC ":O3RFF RFF FRR RRR RFF RFF";
270 MUSIC " O6E1 C<E E2&D A2&G C1 RR";
280 MUSIC ":O3CGG CGG A2. G2. CGG CRR";
290 MUSIC ":O3REE REE F2. F2. REE"

```

## MMLの機能である

では次に MML の機能を説明する。

主な機能は MZ-2500 の BASIC などとほぼ同じであるが、若干機能を削ってある。しかし、実害はないはずである（と突っ張る）。削られた機能のうち一番ナニなのが「連符」



の処理である。MZ-2500 では、

PLAY "{CDE} 4 "

などとすることによって3連符(n連符も可能)が使えたのであるが、こっちはこの機能は使えない。ただしちゃんと逃げ道は用意しておいたので、連符を使いたい人は後で紹介する別の方法を使っていたきたい。

さて機能をまとめたのが表 11-9 である。一つ大事な点は、音長の指定方法である。HuBASIC では 0 が 32 分音符、1 が 16 分音符、……、9 が全音符となっていたが、ここでは MZ-2500 と同じく、32 が 32 分音符、16 が 16 分音符、……、1 が全音符としてある。なお MZ-2500 にはないが 0 は全音符の 2 倍の長さである。それから、「.」を付けることによって音長は 1.5 倍になる。すなわち「C4.」は符点 4 分音符なわけだ。

表 11-9 MML の機能表

文 字	機 能
C-G, A, B	音階
+, #	音階を半音上げる
-	音階を半音下げる
R ℓ	休符 (ℓ: 0-32)
.	音長を 1.5 倍にする
Nx	音階コード x による音の発生 (x: 0-95)
Qn	音の出る時間比率の設定 (n: 1-8)
&(&+)	前後の音をつなぐ(タイ)注)
L ℓ	省略時の音長指定 (ℓ: 0-32)
Ts	テンポ指定 (s: 30-255)
On	オクターブ指定 (n: 1-8)
>	オクターブを一つ上げる
<	オクターブを一つ下げる
Vn	音量設定 (n: 0-127 もしくは 0-15)
In	FM 音源の音色設定 (n: 1-40)
Yr, d	レジスタ r にデータ d をセット
W ℓ	ℓ の長さだけ状態を維持する (ℓ: 1-32)
数値の指定方法 1) "C16" など 2) 音長を直接カウンタ値で指定する場合に限り "C@ 256" など 注) C2&+C4= C2. などとなる	

次に指摘しておくのが音階の指定方法である。HuBASIC では「#C」などで半音上の音を指定していたが、ここでは「C#」もしくは「C+」である。半音下の音は「C-」などである。

あと覚えておくべきことは「Y」によるレジスタへの直接書き込みである。チャンネル 0～7 であれば OPM, 8～10 であれば PSG のレジスタに書き込まれる。

また、1 オクターブ上下の指定であるが、「>」が上がる、「<」が下がるとなっている。それから、音長（および一般的な数値の指定）には 2 とおりある。

- 1) 「C16」, 「T120」, 「V15」のように、素直に書く方法。このとき数値は 0～255 の範囲である
- 2) 「C@512」のように「@」の後に続けて数値を書く。この場合の範囲は 0～65535 で



ある。

3)の方法は連符の処理のために付けたものである。表 11-10 を見ていただきたい。これは各音長の長さの内訳である。たとえば「C4」は「C@256」としてもまったく同じ長さの音なのである。よって4分音符を3個に分けた3連符を鳴らしたい場合は $256=85+85+86$ より、「C@85 C@85 C@86」とすればよいのである。最後だけちょっと長めだが、まあ人間の耳には分かるまい。

表 11-10 音調の内訳

音符長	カウンタ	音符長	カウンタ
0	2048	17	60
1	1024	18	56
2	512	19	53
3	341	20	51
4	256	21	48
5	204	22	46
6	170	23	44
7	146	24	42
8	128	25	40
9	113	26	39
10	102	27	37
11	93	28	36
12	85	29	35
13	78	30	34
14	73	31	33
15	68	32	32
16	64		

そして私がタコなためにしてしまったことであるが、「タイ」の指定に「&」と「&+」の2とおりがあ



などという符は、「G4&G4」となり、これは「G2」と同じであるべきだったのに、私はこ

&=直前の音符を Q8 で鳴らすだけ

だと思ってしまったのである。これは非常にまずかった。で、「&+」を導入して逃げた。これにより、



= 「C4&+C4」= 「C2」





= 「C4&D4」

とすればよいことになる。だが、やはり手抜きは手抜きなので、

「C4&+D4」= 「C4&+C4」= 「C2」

だったりする。少々面倒であるが、「&」と「&+」を使い分けることで我慢していただきたい。

てなわけでリスト 11-6 のサンプルプログラムを見ていただきたい。基本的な使い方を説明すると、まずは演奏に先立って「TEMPO 0」を実行すること。それから、「MUSIC A\$;:MUSIC B\$」= 「MUSIC A\$+B\$」である。このセミコロンはなかなか強力で、これにより 1 パートが 255 文字以内というマヌケな制限が消える。なお演奏を止めるには **CTRL** + **D** である。

では動作確認である。縁起もんだから馬鹿丁寧にやる。

- 1) NEW ON &HE000 を実行。
- 2) LOADM "MML.BIN"
- 3) 「VIP」のディスクから、LOADM "CHOICED VOICE.VTD", &HB000 で、音色のデータをロードする（もちろん VIP で作ったデータならほかのものでもかまわない）。
- 4) CALL &HA8B0 を実行。

これで MML が使えるようになるのである。ちなみに B000<sub>H</sub>~B18F<sub>H</sub>には「音色名」が入っており、リスト 11-7 を実行するとその一覧表を見ることができる。

リスト 11-7 音色名の表示

```
100 FOR I=1 TO 20:J=I+20
110 PRINT #0,I,MEM$(&HB000+(I-1)*10,10),
120 PRINT #0,J,MEM$(&HB000+(J-1)*10,10)
130 NEXT
```

ところで、ちゃんと動くことが分かったら、次には BASIC に組み込んでしまいたいところであるが、面倒になったのでそこまではやってない。私がやったのは、「Start up.Bas」の最後に実行される NEW を NEW ON &HB800 にして、自動的に機械語領域を確保するようにしたことと、

```
100 LOADM "MML.BIN"
110 LOADM "CHOICED VOICE.VTD",&HB000
120 CALL &HA8B0
```

という小さなプログラムをディスクに用意しておくようにしたことだけである。こうしておけば BASIC が立ち上がった後で、上記のプログラムを RUN するだけで MML が使えるようになる。

念のために繰り返すが、音色データは LOADM "(ファイル名)",&HB000 でロードアドレスを指定するのを忘れないように。さもないと BASIC を壊して暴走するのである。



## おまけである

表 11-11 に VIP の音色データのフォーマットを示す。この章で作った MML では、30 バイト目の KEY TRANSPOSE と 35 バイト目の SYC/LFO DELEY は使われていない。また 31, 32, 33, 34 バイト目のデータは、最後の I コマンドで指定されたものが有効になる。よって必要があるならば、実行される I コマンドの順を変えたり、Y コマンドを使うこと。ただ、どうやら LFO に関しては無視してもかまわないようである（自信はないが）。

## 解説するものである

最後に音量の設定についてちょっとだけ説明しておく。FM 音源で音量を変えるには、「アルゴリズム (ALG) を見て、キャリアの TL だけを変える」のである。いままで説明しなかったのだが、オペレータにはキャリアとモジュラの 2 とおりがあるのだ。アルゴリズム 0～3 ではキャリアは OP4 だけ、アルゴリズム 4 では OP4 と OP2 の二つ、アルゴリズム 5, 6 では OP4, OP3, OP2 の三つ、アルゴリズム 7 ではすべてがキャリアとなっている。図 11-5 を見れば分かるだろうが、キャリアとはすなわち最終段にある（他のオペレータを変調しない）オペレータのことである。

実際にリスト 11-4 の AF61<sub>H</sub>～AFBB<sub>H</sub> でどんなことをやっているのかというと、

- 1) 音色データの TL の所をサーチして、最小の TL (つまり最大の音量) になっているキャリアを見つける。
- 2) 1 で見つけたキャリアの TL を V コマンドで指定された値にするための差を D とする ( $D=127-V-TL$  :  $TL=0$  なら  $V=127$  であることに注意)。
- 3) すべてのキャリアの TL に D を加えて、OPM に書き込む。

という具合である。

注意深く見るとあちこちに手抜きが発見できるがあまり堅いことは言わないでいただきたい。



表 11-11 VIP の音色データのフォーマット

0000 <sub>H</sub>	音色名 10文字×40
0190 <sub>H</sub>	
072F <sub>H</sub>	
	パラメータ 36バイト×40

パラメータの内訳

No.	OP	bit								
		7	6	5	4	3	2	1	0	
0		RL		FB			ALG			
1				PMS					AMS	
2	1			Detune1					Phase Multiply	
3	2									
4	3									
5	4									
6	1			Total Level Attenuation						
7	2									
8	3									
9	4									
10	1	Key Scaling				Attack Rate Speed				
11	2									
12	3									
13	4									
14	1	AMS-EN			1st Decay Rate Speed					
15	2									
16	3									
17	4									
18	1	Detune 2				2nd Decay Rate Speed				
19	2									
20	3									
21	4									
22	1	1st Decay Level				Release Rate				
23	2									
24	3									
25	4									
26	1	Key Scale Table,				Key Scale Depth				
27	2									
28	3									
29	4									
30		Key Transpose								
31		LFO Frequency								
32		1	PMD							
33		0	AMD							
34									W/F	
35		SYC	LFO Deley							







第

12

章

# カラーイメージボード



カラーイメージボードで取り込むのである



## 第12章 カラーイメージボードで 取り込むのである……………

この章で取り上げるカラーイメージボードには CZ-8BV1 と CZ-8BV2 の二つがあるわけだ。BV1 と BV2 の主な違いは、BV2 では、スクランブル（ハーフトーン）回路のモードが増えたことと、ハイスピードモノクロなどという技が付いたこと、それに 320/640 ドット、それぞれに対応するスクランブルモードが付いたことなどである。それはともかく、論より証拠である。さっさと表 12-1, 12-2 を見ていただきたい。

まず表 12-1 の CZ-8BV1 である。注意すべき点は、コマンド 10<sub>H</sub> は 1/1 画面モード、リセットコマンド、スクランブル ON の三つの機能を兼ねているということである。よって、1/4 画面および 1/16 画面のときに、

### スクランブル OFF → ON

に切り換えるなどするには、10<sub>H</sub> を出してスクランブルを ON にした後で、もう一度 20<sub>H</sub> か 40<sub>H</sub> を出力して、画面モードを設定し直さなければならない。また、スクランブル OFF の状態のままで、画面モードを 1/4 → 1/1 とするときなどは、1/1 画面モードにすると同時にスクランブルが ON になってしまうので、次に 80<sub>H</sub> を出力して、スクランブルを OFF にしなければならない。

表 12-1 CZ-8BV1

0800 <sub>H</sub>	カラーイメージボードコントロール	OUT
0801 <sub>H</sub>	画像データ読み込み	IN

### ●コマンド

10 <sub>H</sub> <sup>1)</sup>	1/1画面モード+リセット+スクランブルON
20 <sub>H</sub> <sup>1)</sup>	1/4画面モード
40 <sub>H</sub> <sup>1)</sup>	1/16画面モード
80 <sub>H</sub> <sup>1)</sup>	スクランブル回路OFF
01 <sub>H</sub>	次の1ラインを読み始める
02 <sub>H</sub>	次のプレーン(色)を読み始める (B→R→G→B→……)
04 <sub>H</sub> <sup>2)</sup>	コンピュータアクセスモード (バッファを読み始める)
08 <sub>H</sub> <sup>2)</sup>	ビデオ信号入力モード (ビデオ信号をデジタイズし続ける)

注1: 1) および2) のコマンドを出力した後は、最低16.7msのウェイトを取ること（余裕を持って17ms以上が望ましい）。ただし、1) の後に1) のコマンドが続くときならウェイトは必要ない。

注2: コマンド01<sub>H</sub>を出力後、2ms以内に1ラインのデータを転送し終わること。また、このコマンド01<sub>H</sub>を連続して出力するときは、6μs以上の遅延ウェイトを取ること。

注3: 1/4, 1/16画面モードは横方向だけの縮小であるから、縦方向はソフトウェアで「間引く」必要がある。

次に表 12-2 が CZ-8BV2 である。I/O アドレスが+2して、コマンドが四つ増えていることに注目。ここで注意しておかなければならないのが、CZ-8BV2 は、リセット時にスク



ランブルが「モード2」になっているということである。すなわち、CZ-8BV2とはちょっと違うスクランブルをするのである。具体的にどう違うかというと、はっきり言ってモード2の方がより自然な中間色を表現しているのである。モード1では斜めの縞がどうしても目立ったわけであるが、モード2ではそういうことは少なくなっているのである。また、モード2のとき(だけ)はさらにスクランブルの仕方で、320/640ドットモードを選択できるようになっている。これは「スクランブルの仕方」なのである。もともとは640ドットモードなのであるが、もしも本体の画面モードが320ドット(WIDTH 40)だった場合は、320ドットモードのスクランブルの方がより自然に見えるようである。

表 12-2 CZ-8BV2

0802H	カラーイメージボードコントロール	OUT
0803H	画像データ読み込み	IN

● コマンド

10H <sup>1)</sup>	1/1画面モード+リセット+スクランブルON
20H <sup>1)</sup>	1/4画面モード
40H <sup>1)</sup>	1/16画面モード
80H <sup>1)</sup>	スクランブル回路OFF
01H	次の1ラインを読み始める
02H	次のプレーン(色)を読み始める (B→R→G→B→……)
04H <sup>2)</sup>	コンピュータアクセスモード (バッファを読み始める)
08H <sup>2)</sup>	ビデオ信号入力モード (ビデオ信号をデジタイズし続ける)
81H	スクランブルをモード1 (BV1と同じ) にする
83H	320ドットモードのスクランブル
84H	640ドットモードのスクランブル
85H	ハイスピードモノクロモード

注1: CZ-8BV2 では、リセット直後のスクランブルモードは「モード2の640ドットモード」である。よってCZ-8BV1と同じ動作をさせるためには、コマンド81Hを実行する必要がある。

注2: 320/640ドットモードはモード2のときだけ意味を持つ(モード1では両者は同じ)。

## 実践するのである

さっさとサンプルを出してしまうのである。リスト 12-1, 12-2, 12-3, 12-4 である。リスト 12-1 は X1 用, リスト 12-2 は機械語部分, リスト 12-3 は turbo 用, リスト 12-4 は機械語部分である。リスト 12-1, 12-3 とともに, 一般的な変更箇所は,

- 130 行の I/O アドレス
- 170~190 行のモード設定

となっている。CZ-8BV1 を使っているならば, 130 行を,

```
130 CP=&H800:DP=&H801
```

としていただきたい。170~190 行は画面の大きさの選択である。「'」(REM) で殺してない行が選択されるわけだ。おっと, 今気が付いたがこの場合はどちらも WIDTH 80 のモードで走らせていただきたい。



リスト 12-1 X1 用画像取り込みプログラム(遅い)

```

100 'FOR X1
110 INIT:CLS4:DEFINT B-Z
120 CLEAR &HE000
130 CP=&H802:DP=&H803
140 MEM$(&HE000,16)=HEXCHR$("EB 4E 23 46 16 50 D9 01 03 08 ED 78 D9 ED 79 03")
150 MEM$(&HE010,15)=HEXCHR$("15 D9 20 F6 C9")
160 DEFUSR0=&HE000
170 XS=80:YS=200:MD=&H10:SL=1 : '640*200
180 'XS=40:YS=96 :MD=&H20:SL=2 : '320*96
190 'XS=20:YS=48 :MD=&H40:SL=4 : '160*48
200 MEM$(&HE008,2)=MKI$(DP)
210 POKE &HE005,XS
220 OUT CP,&H10 : 'RESET
230 OUT CP,MD : '1/1,1/4,1/16 MODE
240 '
250 OUT CP,&H4 : 'COMP ACCESS (LATCH)
260 PAUSE 1
270 ADR=&H4000:GOSUB"IS"
280 ADR=&H8000:GOSUB"IS"
290 ADR=&HC000:GOSUB"IS"
300 OUT CP,&H8 : 'VIDEO ACCESS
310 PAUSE 1
320 GOTO 250
330 END
340 '
350 LABEL"IS"
360 OUT CP,&H2 : 'RGB
370 FOR I=1 TO 24
380 OUT CP,1 : 'SET NEXT 1 LINE
390 NEXT
400 FOR Y=1 TO YS
410 D=INP(DP):D=INP(DP):D=INP(DP): 'SKIP 3
420 D$=USR0(MKI$(ADR))
430 ADR=ADR+&H800
440 IF (ADR AND &H3800)=0 THEN ADR=ADR-(&H4000-80)
450 FOR I=1 TO SL
460 OUT CP,1: 'PAUSE 0 : 'SET NEXT 1 LINE
470 NEXT
480 NEXT
490 RETURN

```

リスト 12-2 カラーイメージボード→G-RAMへ1ライン転送

			.Z80
			.PHASE 0E000H ;OR ANY PLACE
		;	
E000	EB		EX DE,HL
E001	4E		LD C,(HL)
E002	23		INC HL
E003	46		LD B,(HL) ;BC=VRAM ADDR.
E004	16 50		LD D,80 ;XS
E006	D9		EXX
E007	01 0803		LD BC,0803H ;DPORT
E00A	ED 78	LOOP:	IN A,(C) ;GET 1 BYTE
E00C	D9		EXX
E00D	ED 79		OUT (C),A ;TO VRAM
E00F	03		INC BC ;INC VRAM ADDR.
E010	15		DEC D ;DEC COUNTER
E011	D9		EXX
E012	20 F6		JR NZ,LOOP
		;	
E014	C9		RET
		;	
			END

リスト 12-3 turbo 用画像取り込みプログラム(遅い)

```

100 'FOR turbo
110 INIT:CLS4:DEFINT B-Z
120 CLEAR &HE000

```



```

130 CP=&H802:DP=&H803
140 MEM$(&HE000,12)=HEXCHR$("78 EB 01 80 1F 04 ED A3 3D 20 FA C9")
150 '
160 DEFUSR0=&HE000
170 XS=80:YS=200:MD=&H10:SL=1 : '640*200
180 'XS=40:YS=96 :MD=&H20:SL=2 : '320*96
190 'XS=20:YS=48 :MD=&H40:SL=4 : '160*48
200 OUT CP,&H10 : 'RESET
210 OUT CP,MD : '1/1,1/4,1/16 MODE
220 DMA$=DMA$+CHR$(&B1111001)+MKI$(&H4000)+MKI$(XS-1)
230 DMA$=DMA$+CHR$(&B11100,&B101000,&B11001101)
240 DMA$=DMA$+MKI$(DP)+CHR$(&B10011010)
250 D$=USR0(DMA$)
260 PAUSE 1
270 '
280 DMA$=CHR$(&B11001)+MKI$(0)+HEXCHR$("CF 87")
290 OUT CP,&H4 : 'COMP ACCESS (LATCH)
300 PAUSE 1
310 ADR=&H4000:GOSUB"IS"
320 ADR=&H8000:GOSUB"IS"
330 ADR=&HC000:GOSUB"IS"
340 OUT CP,&H8 : 'VIDEO ACCESS
350 PAUSE 1
360 GOTO 290
370 END
380 '
390 LABEL"IS"
400 OUT CP,&H2 : 'RGB
410 FOR I=1 TO 24
420 OUT CP,1 : 'SET NEXT 1 LINE
430 NEXT
440 FOR Y=1 TO YS
450 D=INP(DP):D=INP(DP):D=INP(DP): 'SKIP 3
460 MID$(DMA$,2,2)=MKI$(ADR)
470 D$=USR0(DMA$)
480 ADR=ADR+&H800
490 IF (ADR AND &H3800)=0 THEN ADR=ADR-(&H4000-80)
500 FOR I=1 TO SL
510 OUT CP,1: 'PAUSE 0 : 'SET NEXT 1 LINE
520 NEXT
530 NEXT
540 RETURN

```

リスト 12-4 文字列→DMA 転送

		.Z80	
		.PHASE 0E000H ;OR ANY PLACE	
		;	
		;DE=ADDRESS TO DATA,B=COUNTER	
E000	78	LD	A,B
E001	EB	EX	DE,HL ;HL POINTS DATA
E002	01 1F80	LD	BC,1F80H ;DMA I/O ADDR.
E005	04	LOOP: INC	B
E006	ED A3	OUTI	
E008	3D	DEC	A
E009	20 FA	JR	NZ,LOOP
E00B	C9	RET	
		;	
		END	

プログラムの説明に入る。まずはリスト 12-1 である。大事なものは 140, 150 行で機械語ルーチンを用意している点である。この機械語ルーチンは手抜きしてあるので, 200, 210 行で一部を書き換えたりしていてあまりよくない。

さて, まずは 220 行である。これはコントロールポートに 10<sub>H</sub>を出力しているわけだ。これが何かと見るならば, 表 12-1 より明らかなように, 「1/1 画面モード+リセット+スク



ランプ ON」なわけだ。その次の 230 行では再びコントロールポートに MD を出力している。これは何かと見るならば、結局 MD は 10<sub>H</sub>, 20<sub>H</sub>, 40<sub>H</sub>のうちのどれかなのである。MD=10<sub>H</sub>の場合は、先程と同じ命令を繰り返すことになるわけだから、ま、無駄ということになる。そこで MD=20<sub>H</sub>の場合を説明するのである。

コントロールポートに 20<sub>H</sub>が出力される前は、カラーイメージボードは 1/1 画面モードでシャコシャコとビデオ入力をデジタイズしていたわけである。すなわち、カラーイメージボードはビデオ画面を 640 ドット×200 ドット=80 バイト×200 ライン（実を言うと本当はもっと大きいのだが）のデータに変換していたのである。そこへ 20<sub>H</sub>というコマンドが来たわけである。そうすると、カラーイメージボードはそれまでの 640×200 をやめて、ビデオ信号を 320 ドット×200 ドット=40 バイト×200 ラインのデータに変換するようになるのだ。縦の方は 200 ラインのまま変わらないということに注意。もしもこれが 20<sub>H</sub>でなく 40<sub>H</sub>であったなら、160 ドット×200 ドット=20 バイト×200 ラインとなる（ここでは簡単に説明するために「200 ライン」と書いてあるが、実はカラーイメージボードはもっと多くのラインを取り込んでいるのである。実際に表示するのはそのうちの 200 ラインということになる）。

これらのデータを変に潰れたり、伸びたりせずに表示させるには、コンピュータの画面モード（WIDTH）との関係で表 12-3 のようにすることになる。

表 12-3 カラーイメージボードと本体の画面モード

	WIDTH 40	WIDTH 80
1/1=10 <sub>H</sub>	40バイト×200ラインをそのまま G-RAM へ（画面全体に表示）	80バイト×200ラインをそのまま G-RAM へ（画面全体に表示）
1/4=20 <sub>H</sub>		40バイト×200ラインを1ラインおきに G-RAM へ（画面の1/4に表示）
1/16=30 <sub>H</sub>	20バイト×200ラインを1ラインおきに G-RAM へ（画面の1/4に表示）	20バイト×200ラインを3ラインおきに G-RAM へ（画面の1/16に表示）

さてここからが佳境である。リスト 12-1 の 250 行でコントロールポートに 04<sub>H</sub>を出力している。これは表 12-1 にもあるように、「コンピュータアクセスモード」なのである。早い話が、カラーイメージボードはこのコマンドを受け取ると、それまでシャカシャカとデジタイズしていたのをやめて、「よっしゃよっしゃ、デジタイズしたデータが欲しいんだな。ようし、イキのいいやつをデジタイズして渡してやらあ。受け取りな、ほうら」となるわけである。すなわち、カラーイメージボードは、自分の持っているバッファを一杯にした後で「データ転送モード」に入るのである（表 12-1 の注 1 にもあるように、バッファが一杯になるまで、最悪の場合 17 ms 待たなければならない）。

その次の 270~290 行では青、赤、緑の順でカラーイメージボードから G-RAM に転送している。

というわけで、350 行からのサブルーチン “IS” の説明である。

まずは 360 行で「2」を OUT している。これは言わずと知れた B → R → G → B……な



わけだ。次に 370～390 行のループであるが、これは 24 行の空読みである。先程書いたように、カラーイメージボードは実際は 200 ラインよりも多くのライン数を取り込んでいるのだ。それで、ちょうどおいしい所だけを表示するために、最初の方は空読みして捨てているのである。それから 410 行である。今度は 3 バイトの空読みである。何でこうするのかというと、それはカラーイメージボードのハードがそうなっているからである。試しに 410 行を削除してみると、表示が少し右にずれるのが分かるだろう。後は 420 行で 1 ライン転送し、430 行、440 行で G-RAM アドレスを 1 ライン下げ、450～470 行で（カラーイメージボード側の）ラインのスキップをやっている。

以上で一とおり説明は終わりである。170～190 行で注釈にしてある行を復活して大ききの違う画面を表示したり、YS の値を大きくしたり、SL の値をいじったりして楽しんでいただきたい。

リスト 12-3 の turbo 版は、データの転送に DMA を使っていること以外は変わったことはないなので、説明は省略する。

## 気分は近未来である

さて、ここからぼちぼちフィニッシュに持ち込むのである。そして一体何をやるかというと、「BASIC+動画面のウィンドウ」なのである。

種を明かすと、CTC の割り込みを使って、BASIC と画像取り込みが並列動作してしまうのである。言葉で言うと簡単に聞こえてしまうが、これはちょっと他機種ではまねができないワザであろう。BASIC がちゃんと動いている同じ画面で、1/16（もしくは 1/4）の画面がリアルタイムで動いているのである。これを見てむむむとうならない奴はいないであろう。

で、このよーに BASIC+画像取り込みが（一応）マルチタスクすると、どういうメリットがあるかというと、

- 1) 他機種のユーザーに自慢できる
- 2) 他機種のユーザーが驚く
- 3) 他機種のユーザーが落ち込む
- 4) 他機種のユーザーが寝返る
- 5) テレビを見ながらプログラムや、機械語のダンプリストを入力できる（スーパーインポーズよりずっと目触りがよい）
- 6) 近未来の気分を味わえる

となっている。とにもかくにも、8 色ながらも「動画面のウィンドウ」なのである。こんな SF っぽいことを、8 ビットの turbo（と CTC を持つ X1）でできてしまうのだだっ！と逆上しつつ、プログラムの説明になだれ込むのであった。

リスト 12-5 が X1 用のアセンブルリスト、リスト 12-6 がそのダンプリスト（CRC チェックサム付き：付録 A を参照）、リスト 12-7 が turbo 用のアセンブルリスト、リスト 12-8 がその CRC チェックサム付きダンプリストとなっている。



そして使い方はリスト 12-9 ~ 12-12 である。

順に説明しよう。

おのこののダンプリストを打ち込み、リスト 12-6 は、

SAVEM "BV.CPU", &HEF00, &HEFC6

でセーブする。

リスト 12-8 ならば、

SAVEM "BV.DMA", &HEF00, &HEFEE

である。チェックサム (および CRC) の確認を確実にすること。

#### リスト 12-5 X1 用画像取り込みプログラム

		.Z80	
		.PHASE	0EF00H
		;	
		COMMANDS	10H RESET, 1/1
			20H 1/4
			40H 1/16
			80H HALF TONE OFF
			01H NEXT 1 LINE
			02H NEXT PLANE(B->R->G->B..)
			04H COMP ACCESS
			08H VIDEO ACCESS
		;	
		CZ-8BV2	
			81H HALF TONE=MODE1
			83H X=320 MODE
			84H X=640 MODE
			85H MONO COLOR MODE
EF00	50	XS: DB	80 ;X SIZE
EF01	C8	YS: DB	200 ;Y SIZE
EF02	01	UPPASS: DB	1 ;UP PASS LINES
EF03	01	LTPASS: DB	1 ;LEFT PASS BYTES
EF04	01	STIME: DB	1 ;SKIP LINES
EF05	10 00	MODES: DB	10H, 00H
EF07	00 00 00 00	DB	00H, 00H, 00H, 00H, 00H
EF0B	00		
EF0C	0802	CPORTA: DW	0802H ;OR 800H
EF0E	4000	TOPA: DW	4000H
		;	
		=====	
EF10	CD EFAC	START: CALL	INIT ;INIT BV
EF13	F3	ENT: DI	
EF14	F5	PUSH	AF
EF15	C5	PUSH	BC
EF16	D5	PUSH	DE
EF17	E5	PUSH	HL
EF18	D9	EXX	
EF19	C5	PUSH	BC
EF1A	D5	PUSH	DE
EF1B	E5	PUSH	HL ;SAVE REGISTERS
EF1C	ED 4B EF0C	LD	BC, (CPORTA)
EF20	3E 04	LD	A, 04H ;COMP ACCESS
EF22	ED 79	OUT	(C), A ;LATCH
EF24	D9	EXX	;BC'=(CPORTA)
EF25	CD EFBE	CALL	DELAY
EF28	2A EF0E	LD	HL, (TOPA) ;VRAM ADDRESS
EF2B	E5	PUSH	HL
EF2C	CD EF52	CALL	IMAGES ;BLUE
EF2F	E1	POP	HL
EF30	E5	PUSH	HL
EF31	CB B4	RES	6, H
EF33	CB FC	SET	7, H
EF35	CD EF52	CALL	IMAGES ;RED



EF38	E1	POP	HL	
EF39	CB FC	SET	7,H	
EF3B	CD EF52	CALL	IMAGES	;GREEN
EF3E	3E 08	LD	A,08H	;VIDEO ACCESS
EF40	D9	EXX		;BC'=(CPORTA)
EF41	ED 79	OUT	(C),A	
EF43	D9	EXX		
EF44	CD EFBE	CALL	DELAY	
EF47	E1	POP	HL	
EF48	D1	POP	DE	
EF49	C1	POP	BC	
EF4A	D9	EXX		
EF4B	E1	POP	HL	
EF4C	D1	POP	DE	
EF4D	C1	POP	BC	
EF4E	F1	POP	AF	;LOAD REGISTERS
EF4F	FB	EI		
EF50	C9	RET		;OR RETI
EF51	00	NOP		
;=====				
EF52	3E 02	IMAGES: LD	A,02H	;NEXT PLANE
EF54	D9	EXX		;BC'=(CPORTA)
EF55	ED 79	OUT	(C),A	
EF57	D9	EXX		
EF58	CD EF91	CALL	DUMYV	;SKIP LINES
EF5B	3A EF01	LD	A,(YS)	;LINE COUNT
EF5E	5F	LD	E,A	;E=COUNTER
EF5F	CD EF9F	IMAY: CALL	DUMYH	;SKIP BYTES
EF62	44	LD	B,H	
EF63	4D	LD	C,L	;COPY ADDRESS
EF64	3A EF00	LD	A,(XS)	
EF67	57	LD	D,A	;COUNTER
EF68	D9	EXX		;BC'=(CPORTA)
EF69	03	INC	BC	;BC'=(DPORTA)
EF6A	ED 78	POO: IN	A,(C)	;READ 1 BYTE
EF6C	D9	EXX		
EF6D	ED 79	OUT	(C),A	;TO VRAM
EF6F	03	INC	BC	;INC VRAM ADDR.
EF70	15	DEC	D	;DEC COUNTER
EF71	D9	EXX		
EF72	C2 EF6A	JP	NZ,POO	
EF75	0B	DEC	BC	;BC'=(CPORTA)
EF76	D9	EXX		
EF77	3A EF04	LD	A,(STIME)	
EF7A	CD EF94	CALL	NEXTL	;SKIP LINES
EF7D	3E 08	LD	A,8	;DOWN 1 LINE
EF7F	84	ADD	A,H	
EF80	67	LD	H,A	
EF81	E6 38	AND	38H	
EF83	C2 EF8C	JP	NZ,IMANEX	
EF86	01 3FB0	LD	BC,4000H-80	
EF89	B7	OR	A	
EF8A	ED 42	SBC	HL,BC	
EF8C	1D	IMANEX: DEC	E	
EF8D	C2 EF5F	JP	NZ,IMAY	;LOOP
EF90	C9	RET		
EF91	3A EF02	;?? LINES PASS		
EF94	D9	DUMYV: LD	A,(UPPASS)	
EF95	16 01	NEXTL: EXX		;BC'=(CPORTA)
EF97	ED 51	LD	D,01H	;1L DOWN
EF99	3D	NEXTL1: OUT	(C),D	
EF9A	C2 EF97	DEC	A	
EF9D	D9	JP	NZ,NEXTL1	
EF9E	C9	EXX		
		RET		
EF9F	D9	;?? BYTES SKIP		
EFA0	0C	DUMYH: EXX		;BC'=(CPORTA)
EFA1	3A EF03	INC	C	;BC'=(DPORTA)
		LD	A,(LTPASS)	



EFA4	ED 50	DUMYHL: IN	D, (C)
EFA6	3D	DEC	A
EFA7	20 FB	JR	NZ, DUMYHL
EFA9	0D	DEC	C
EFAA	D9	EXX	;BC'=(CPORTA)
EFAB	C9	RET	
;			
EFAC	ED 4B EF0C	INIT: LD	BC, (CPORTA)
EFB0	21 EF05	LD	HL, MODES
EFB3	7E	INIT0: LD	A, (HL)
EFB4	23	INC	HL
EFB5	B7	OR	A
EFB6	C8	RET	Z ;END MODE SET
EFB7	ED 79	OUT	(C), A ;SET BV MODE
EFB9	CD EFBE	CALL	DELAY
EFBC	18 F5	JR	INIT0
;			
EFBE	11 0A37	DELAY: LD	DE, 2615 ;10
EFC1	1B	DELAYL: DEC	DE ;6
EFC2	7A	LD	A, D ;4
EFC3	B3	OR	E ;4
EFC4	20 FB	JR	NZ, DELAYL ;12/7
EFC6	C9	RET	;10
;10+10+(6+4+4+12)*2615-12+7=68005			
;=17.00125ms			
;			
;			
END			

リスト 12-6 「リスト 12-5」のダンプリスト \*BV. CPU\*でセーブ

EF00	50 C8 01 01 01 10 00 00	: 2B	EF78	04 EF CD 94 EF 3E 08 84	: 0D
EF08	00 00 00 00 02 08 00 40	: 4A	-----		
EF10	CD AC EF F3 F5 C5 D5 E5	: CF	SUM:	F5 DA 90 88 42 94 1F 75 92ED	
EF18	D9 C5 D5 E5 ED 4B 0C EF	: 8B	-----		
EF20	3E 04 ED 79 D9 CD BE EF	: FB	EF80	67 E6 38 C2 8C EF 01 B0	: 73
EF28	2A 0E EF E5 CD 52 EF E1	: FB	EF88	3F B7 ED 42 1D C2 5F EF	: 52
EF30	E5 CB B4 CB FC CD 52 EF	: 39	EF90	C9 3A 02 EF D9 16 01 ED	: D1
EF38	E1 CB FC CD 52 EF 3E 08	: FC	EF98	51 3D C2 97 EF D9 C9 D9	: 51
EF40	D9 ED 79 D9 CD BE EF E1	: 73	EFA0	0C 3A 03 EF ED 50 3D 20	: D2
EF48	D1 C1 D9 E1 D1 C1 F1 FB	: CA	EFA8	FB 0D D9 C9 ED 4B 0C EF	: DD
EF50	C9 00 3E 02 D9 ED 79 D9	: 21	EFB0	21 05 EF 7E 23 B7 C8 ED	: 22
EF58	CD 91 EF 3A 01 EF 5F CD	: A3	EFB8	79 CD BE EF 18 F5 11 37	: 48
EF60	9F EF 44 4D 3A 00 EF 57	: 9F	EFC0	0A 1B 7A B3 20 FB C9	: 36
EF68	D9 03 ED 78 D9 ED 79 03	: 83	-----		
EF70	15 D9 C2 6A EF 0B D9 3A	: 27	SUM:	6B 48 EC 62 A6 E2 15 98 168C	

リスト 12-7 X1 turbo 用画像取り込みプログラム

.Z80			
.PHASE 0EF00H			
1F80	DMA EQU	1F80H	
;			
;COMMANDS			
		10H	RESET, 1/1
		20H	1/4
		40H	1/16
		80H	HALF TONE OFF
		;	
		01H	NEXT 1 LINE
		02H	NEXT PLANE(B->R->G->B..)
		04H	COMP ACCESS
		08H	VIDEO ACCESS
;			
;CZ-8BV2			
		81H	HALF TONE=MODE1
		83H	X=320 MODE
		84H	X=640 MODE
		85H	MONO COLOR MODE
;			
EF00	50	XS: DB	80 ;X SIZE
EF01	C8	YS: DB	200 ;Y SIZE
EF02	01	UPPASS: DB	1 ;UP PASS LINES
EF03	01	LTPASS: DB	1 ;LEFT PASS BYTES
;			



EF04	01	STIME: DB 1 ;SKIP LINES
EF05	10 00	MODES: DB 10H,00H
EF07	00 00 00 00	DB 00H,00H,00H,00H,00H
EF0B	00	
EF0C	0802	CPORTA: DW 0802H ;OR 800H
EF0E	4000	TOPA: DW 4000H
		;
		=====
EF10	CD EFAA	START: CALL INIT ;INIT BV,DMA
EF13	F3	ENT: DI
EF14	F5	PUSH AF
EF15	C5	PUSH BC
EF16	D5	PUSH DE
EF17	E5	PUSH HL
EF18	D9	EXX
EF19	C5	PUSH BC
EF1A	D5	PUSH DE
EF1B	E5	PUSH HL ;SAVE REGISTERS
		;
EF1C	ED 4B EF0C	LD BC,(CPORTA)
EF20	3E 04	LD A,04H ;COMP ACCESS
EF22	ED 79	OUT (C),A ;LATCH
EF24	D9	EXX ;BC'=(CPORTA)
EF25	CD EFE6	CALL DELAY
		;
EF28	2A EF0E	LD HL,(TOPA) ;VRAM ADDRESS
EF2B	E5	PUSH HL
EF2C	CD EF52	CALL IMAGES ;BLUE
		;
EF2F	E1	POP HL
EF30	E5	PUSH HL
EF31	CB B4	RES 6,H
EF33	CB FC	SET 7,H
EF35	CD EF52	CALL IMAGES ;RED
		;
EF38	E1	POP HL
EF39	CB FC	SET 7,H
EF3B	CD EF52	CALL IMAGES ;GREEN
		;
EF3E	3E 08	LD A,08H ;VIDEO ACCESS
EF40	D9	EXX ;BC'=(CPORTA)
EF41	ED 79	OUT (C),A
EF43	D9	EXX
		;
EF44	CD EFE6	CALL DELAY
		;
EF47	E1	POP HL
EF48	D1	POP DE
EF49	C1	POP BC
EF4A	D9	EXX
EF4B	E1	POP HL
EF4C	D1	POP DE
EF4D	C1	POP BC
EF4E	F1	POP AF ;LOAD REGISTERS
		;
EF4F	FB	EI
EF50	C9	RET ;OR RETI
EF51	00	NOP
		=====
		;
EF52	3E 02	IMAGES: LD A,02H ;NEXT PLANE
EF54	D9	EXX ;BC'=(CPORTA)
EF55	ED 79	OUT (C),A
EF57	D9	EXX
		;
EF58	CD EF8F	CALL DUMYV ;SKIP LINES
EF5B	3A EF01	LD A,(YS) ;LINE COUNT
EF5E	5F	LD E,A ;E=COUNTER
EF5F	CD EF9D	IMAY: CALL DUMYH ;SKIP BYTES
EF62	01 1F80	LD BC,DMA ;DMA ADDRESS
EF65	3E 19	LD A,00011001B ;WR0
EF67	ED 79	OUT (C),A
EF69	ED 69	OUT (C),L
EF6B	ED 61	OUT (C),H ;SET VRAM ADDRESS
EF6D	3E CF	LD A,0CFH ;WR6 LOAD
EF6F	ED 79	OUT (C),A
EF71	3E 87	LD A,87H ;WR6 ENABLE
EF73	ED 79	OUT (C),A
		;



294 試験に出る X1



```

;10+10+(6+4+4+12)*2615-12+7=68005
;
;
END

```

リスト 12-8 「リスト 12-7」のダンプリスト“BV. DMA”でセーブ

EF00 50 C8 01 01 01 10 00 00 : 2B	EF80 38 C2 8A EF 01 B0 3F B7 : 1A
EF08 00 00 00 00 02 08 00 40 : 4A	EF88 ED 42 1D C2 5F EF C9 3A : 5F
EF10 CD AA EF F3 F5 C5 D5 E5 : CD	EF90 02 EF D9 16 01 ED 51 3D : 5C
EF18 D9 C5 D5 E5 ED 4B 0C EF : 8B	EF98 C2 95 EF D9 C9 D9 0C 3A : 07
EF20 3E 04 ED 79 D9 CD E6 EF : 23	EFA0 03 EF ED 50 3D 20 FB 0D : 94
EF28 2A 0E EF E5 CD 52 EF E1 : FB	EFA8 D9 C9 ED 4B 0C EF 21 05 : FB
EF30 E5 CB B4 CB FC CD 52 EF : 39	EFB0 EF 7E 23 B7 28 07 ED 79 : DC
EF38 E1 CB FC CD 52 EF 3E 08 : FC	EFB8 CD E6 EF 18 F4 3A 00 EF : D7
EF40 D9 ED 79 D9 CD E6 EF E1 : 9B	EFC0 6F 26 00 2B 22 DE EF 2A : D9
EF48 D1 C1 D9 E1 D1 C1 F1 FB : CA	EFC8 0C EF 23 22 E3 EF 21 DD : 10
EF50 C9 00 3E 02 D9 ED 79 D9 : 21	EFD0 EF 3E 09 01 80 1F 04 ED : C7
EF58 CD 8F EF 3A 01 EF 5F CD : A1	EFD8 A3 3D 20 FA C9 61 ED 1E : 2F
EF60 9D EF 01 80 1F 3E 19 ED : 70	EFE0 1C 28 CD 0C CD 9A 11 37 : CC
EF68 79 ED 69 ED 61 3E CF ED : 17	EFE8 0A 1B 7A B3 20 FB C9 : 36
EF70 79 3E 87 ED 79 3A 04 EF : D1	
EF78 CD 92 EF 3E 08 84 67 E6 : 65	
SUM: C0 C8 B0 5D 52 C0 51 0C 3533	SUM: B4 77 EE 11 CA 97 49 2B F2C9

次に小手しらべである。まず X1 ユーザーはリスト 12-9 を, turbo ユーザーはリスト 12-10 を打ち込む (120 行が 1 行違うだけである)。暴走すると元も子もなくなるのでセーブを忘れないように。

それから RUN すると (あらかじめ機械語部分を LOADM でロードしておいた方がよい), 全画面取り込みが始まるのである。それがうまくいったらこっちのものである。  
**SHIFT** + **BREAK** で止めていただきたい。次に 130~160 行をリスト 12-11, 12-12 のように打ち換えてそれぞれセーブしていただきたい。おのおのが 1/4 画面, 1/16 画面の取り込みを実行するのである。

ここで CZ-8BV2 を持っている人のための腕慣らしである。リスト 12-11 の 150 行にあるコマンドの並びを,

HEXCHR\$("10 20 83.....

としていただきたい。よく観察した後で, BREAK して, 次には,

HEXCHR\$("10 20 84.....

としていただきたい。どうだ, 微妙にスクランブルのパターンが違うであろう。これが 320 ドットと 640 ドットモードの違いである。

次に同じようなことを,

HEXCHR\$("10 20 81.....

と,

HEXCHR\$("10 20 00.....

でもやっていただきたい。これがモード 1 とモード 2 の違いである。確かにモード 2 の方が自然な感じであろう。また, モード 1 では, 320/640 ドットの違いがないということもチェックしていただきたい。さらには申し訳ないのだが, ハイスピードモノクロに関しては自習ということにする。機械語プログラムを適当に書き換えるなどしてみたい。モノ



クロであるから、パレットを替えて、1画面だけ転送すればよいのである。

リスト 12-9 X1 用全画面取り込み

```
100 INIT          : '1/1 MODE
110 CLEAR &HEF00
120 IF MEM$(&HEF10,3)<>HEXCHR$("CD AC EF") THEN LOADM "BV.CPU"
130 MEM$(&HEF00,4)=CHR$(80,200,24,3)
140 MEM$(&HEF04,1)=CHR$(1)
150 MEM$(&HEF05,7)=HEXCHR$("10 00 00 00 00 00 00")
160 MEM$(&HEF0C,4)=MKI$(&H802)+MKI$(&H4000)
170 CALL &HEF10
180 CALL &HEF13:GOTO 180
```

リスト 12-10 X1 turbo 用全画面取り込み

```
100 INIT          : '1/1 MODE
110 CLEAR &HEF00
120 IF MEM$(&HEF10,3)<>HEXCHR$("CD AA EF") THEN LOADM "BV.DMA"
130 MEM$(&HEF00,4)=CHR$(80,200,24,3)
140 MEM$(&HEF04,1)=CHR$(1)
150 MEM$(&HEF05,7)=HEXCHR$("10 00 00 00 00 00 00")
160 MEM$(&HEF0C,4)=MKI$(&H802)+MKI$(&H4000)
170 CALL &HEF10
180 CALL &HEF13:GOTO 180
```

リスト 12-11 4分の1画面取り込み用変更点

```
130 MEM$(&HEF00,4)=CHR$(43,116,1,2)
140 MEM$(&HEF04,1)=CHR$(2)
150 MEM$(&HEF05,7)=HEXCHR$("10 20 00 00 00 00 00")
160 MEM$(&HEF0C,4)=MKI$(&H802)+MKI$(&H4000+37+80*10)
```

リスト 12-12 16分の1画面取り込み用変更点

```
130 MEM$(&HEF00,4)=CHR$(21,58,1,2)
140 MEM$(&HEF04,1)=CHR$(4)
150 MEM$(&HEF05,7)=HEXCHR$("10 40 00 00 00 00 00")
160 MEM$(&HEF0C,4)=MKI$(&H802)+MKI$(&H4500+59+80)
```

さて、ここからがキメである。1/16画面の取り込みプログラム(1/1画面でも1/4画面でもいいが)の180行以降をリスト12-13(turbo用)もしくはリスト12-14(X1用)のように書き換えるのである。そうすればBASICは「OK」とメッセージを出す、画像取り込みは相変わらず行なわれている。すなわちマルチタスク(?)のでき上がりである。ただし、X1用の方は、FM音源ボード(CZ-8BS1)か、立体視ボード(CZ-8RB1)か、RS-232C・マウスボード(CZ-8BM2)が必要である。そして、それぞれの場合において200~220行にあるように「CTC=～」のアドレスを書き換えること。リスト12-14ではFM音源ボードに対応するようになっている。

このプログラムはなかなか趣があるわけだが、欠点は少々キーの反応が悪くなることである。turboの場合はTEMPO文で適当に調節していただきたい(「TEMPO 30」で画



像取り込みが一番遅くなる)。X1の方では、300行と320行でCTCに送っている値(30と125)を大きくすると画像取り込みが遅くなり、小さくすると速くなる。これについては第5章を参照のこと。ところでリスト12-14では190行で「POKE &H61AE, 1」とやっているが、これはBASICを書き換えて、「LIST」命令を実行してもグラフィックがOFFにならないようにしているのである(ただし「FILES」などではOFFになる)。ちなみにturboでは(リスト12-13ではやってないが)、「POKE &H14F9, 1」とすることによって、「LIST」を実行してもグラフィックがOFFにならなくなる。

なお注意しておくが、リスト12-13の200行は「割り込みを禁止せずにベクトルテーブルを書き換えている」ので、とんでもない反則である。やりすぎると暴走する。また、リスト12-13、12-14は「何度もRUNさせてはいけない」のである。つまり、割り込みを禁止せずに割り込み処理ルーチンのワークをいじることになるからである。すなわち、2度目のRUN以降はロシアンルーレットみたいなものである。

また、プログラム中でDELAYというサブルーチンを使っているが、その時間はドブに捨ててしまっているのである。つまり、CTCをうまく使うともっと時間を有効に使え、BASICのキー反応が向上するのである。それらに関しては自由研究としておきたい。

#### リスト 12-13 X1 turbo 用テレビインリスト変更点(turbo BASIC 用)

```
180 TEMPO 80
190 MEM$(&HEF50,2)=HEXCHR$("ED 4D")
200 MEM$(&HF81E,2)=MKI$(&HEF13)
```

#### リスト 12-14 X1 用テレビインリスト変更点(NEW BASIC では動かない)

```
180 MEM$(&HEF50,2)=HEXCHR$("ED 4D")
190 POKE &H61AE,1 : 'IF TAPE BASIC THEN &H68E2
200 CTC=&H704 : 'CZ-8BS1
210 'CTC=&HA04 : 'CZ-8BR1
220 'CTC=&H1FA8 : 'CZ-8BM2
230 OUT CTC+0,3 : 'チャンネル0 RESET
240 OUT CTC+1,3 : 'チャンネル1 RESET
250 OUT CTC+2,3 : 'チャンネル2 RESET
260 OUT CTC+3,3 : 'チャンネル3 RESET
270 '
280 MEM$(&H5E,2) =HEXCHR$("13 EF") : 'ワリコミ TABLE SET
290 '
300 OUT CTC+0,&B100111 : OUT CTC+0,30 : 'チャンネル0 SET
310 OUT CTC+0,&H58 : 'ワリコミベクトル SET
320 OUT CTC+3,&B11000111 : OUT CTC+3,125 : 'チャンネル3 SET
```







第

13

章

# データレコーダ



テープもやってしまうのである



## ■ 第13章

# テープもやってしまうのである・・・

この章ではデータレコーダについてやるのであるが、最初に断っておくと、私はテープには未来がないと思っているのである。確かにテープは安価であるし、X1、MZでは信頼性あってボーレートもまあまあである。しかし、いかんせんディスクにはかなうべくもない。たとえばアセンブラを使おうなどと考えたなら、1バイトの修正に平気で10分以上かかってしまうのである（もちろんそうでないアセンブラもあるが、その場合はアセンブルできるオブジェクトの大きさが制限されたりするのだ）。そのように考えてみるなら、私が「う～む、これはテープのテクニックを極めたとしても、応用は限られてくるなあ」と考えたとしても、もっともな話であろう。

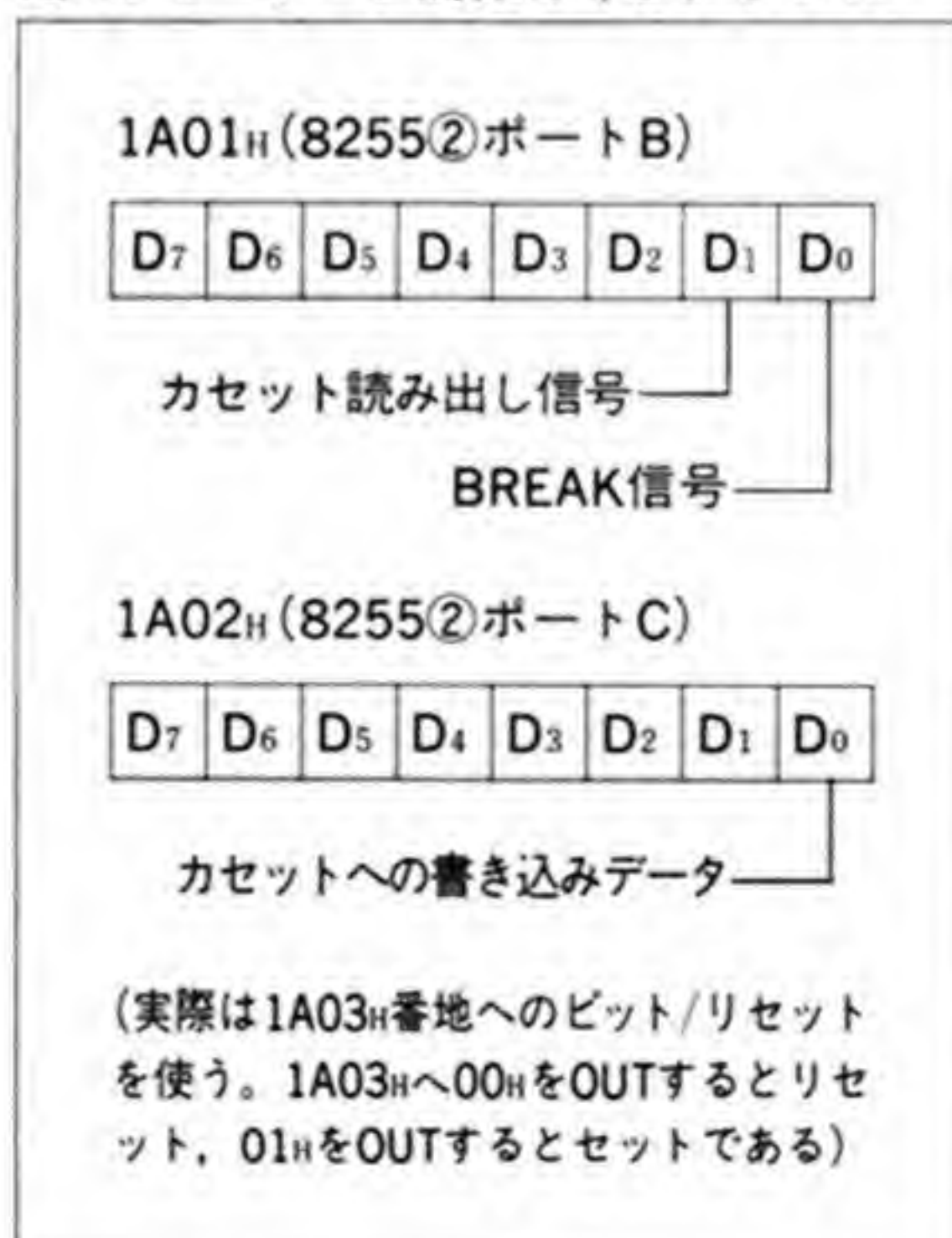
しかしこの世にはまだまだテープユーザーがいることでもあるから、やはりやってしまうのである。

## テープ関係のI/Oアドレスである

早い話が図13-1である。

ここで注意が必要なのは、1A01<sub>H</sub>番地の BREAK 信号である。ここはカセットが動作中に **SHIFT** + **BREAK** (もしくは **CTRL** + **C**) が押されたり、カセットコントロールキーが押されたりして動作がその筋したときなどに Low (= 0) になるのである。よってここをチェックしないと、カセットメカが停止しているにもかかわらず、プログラムの方はぽかんと口を開けてデータを待っているというだらしのないことになってしまうのである。そのようなプログラムはフルロジックコントロール可能な X1 では **御法度** なのであ

図 13-1 テープ関係の I/O アドレス





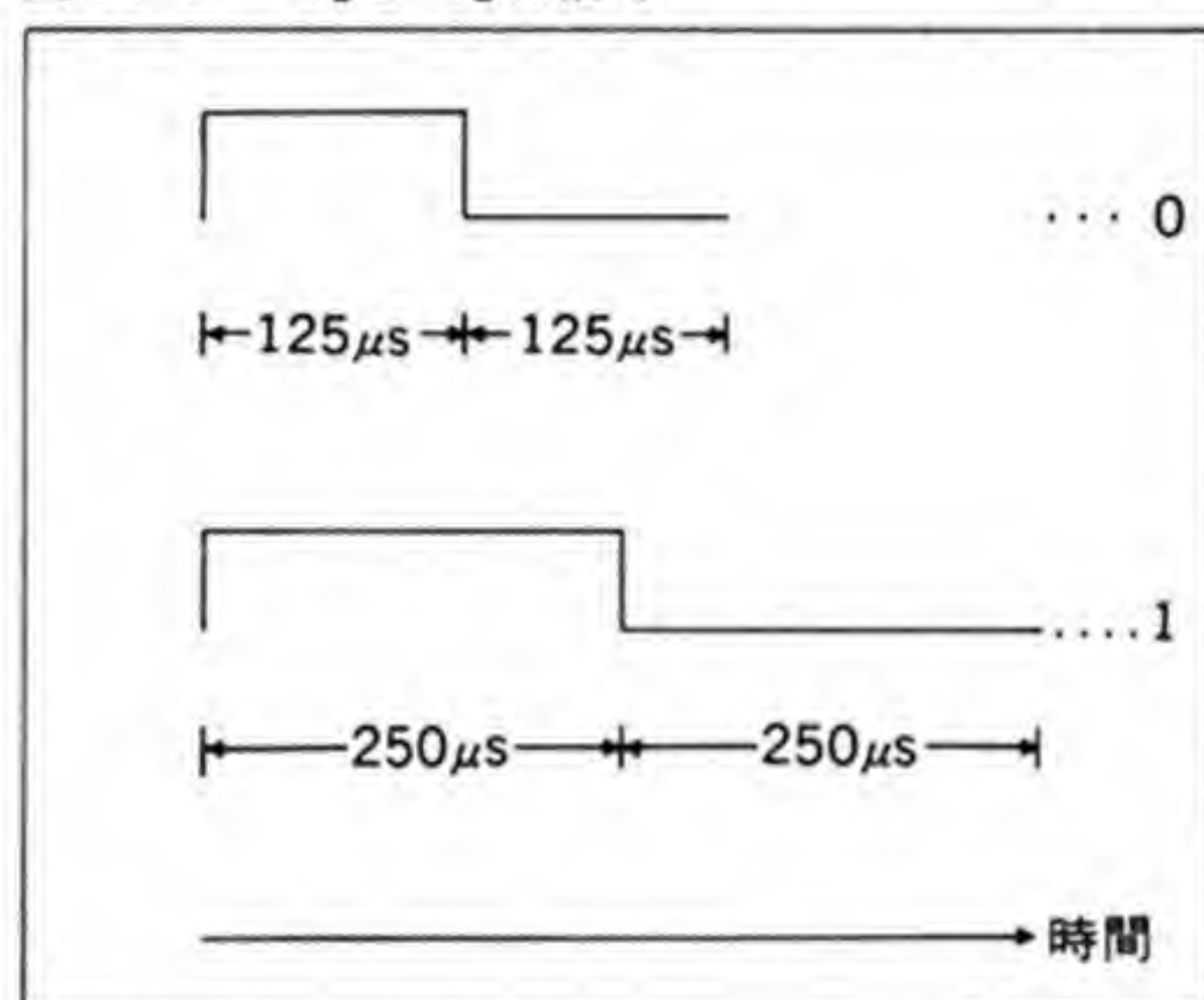
る。心得ていただきたい。

あとっておくべきこととなると、サブ CPU のことがあるが、詳しくは第 4 章を見ていただきたい。

## 0と1である

図 13-2 を見ていただきたい。これらが X1 のテープにおける 0 と 1 である。0 と 1 が記録できるなら、デジタルデータに限れば十分に戦闘可能になる。実際にはこれらのほかに 0 でも 1 でもないものもあるわけだが、それらは単なる「無記録部分」として扱われる。

図 13-2 「0」と「1」の信号



では具体的に 0, 1 を書く方法と読む方法を示すことにする。

まずは書く方である。図 13-2 に示すような信号をテープに記録するわけだから、適当にタイミングを取って「カセットテープへの書き込みデータ」を L → H → L にしてやればよいのである。

次に読む方である。方法はいくらでも考えられるが、主流は、

- 1) 信号の立ち上がり (L → H) をとらえる
- 2) 185 μs 待つ
- 3) 信号を読み取って L なら 0, H なら 1

という手法である。読者の中には「立ち上がりから立ち下がりまでの時間を計ってもいいじゃないか」と考える人もいるだろうが、それはいけないのである。なぜかという、それだとノイズを拾う確率がどどーんと増えてしまうからである。「1回のサンプリング当たりの平均エラーレートがこれこれだとすると、0と1を読み間違える確率があーたらこーたら」というようなことは、若干なのでやめておくが、先程述べた主流の方法で世の中が丸く収まっているのだから、とりあえずはこれでいいのだ。しかし、広い世の中には「うんにゃ！ 私はテープリードエラーで痛い目を見たことがある。読むアルゴリズムを変えれば、あのときのプログラムを回収できるかもしれない！」などと言う人もいるかもしれない。確かにそのとおりであろう。この本は、そのよーな人が自力で問題を解決するという基本姿勢を保ちつつ、さっさと次に行ってしまうのであった。



## テープの記録フォーマットである

「0」と「1」をテープに記録することによって、プログラムやデータを保存するわけであるが、やはり世の常としていきなりファイル名やデータを書くわけにはいかないのである。なぜならば、テープはどこから読み始められるか分からないので、「今読んでいるのはファイル名なのか？ それともデータの途中なのか？」ということの判別できなければならないからである。つまり「はいっ！ もうじき“ファイル名”の始まりですよっ！」とかいう特別なマークが必要なのだ。そんなこんなの形式は各機種、各システムで違ってくるわけであるが、図 13-3 が一番大事な X1 HuBASIC でのフォーマットである。MZ でのフォーマットも図 13-4 に示しておいた。これらは「Oh! MZ」84 年 7 月号に掲載されていたものである。

図 13-3 X1 HuBASIC でのテープフォーマット

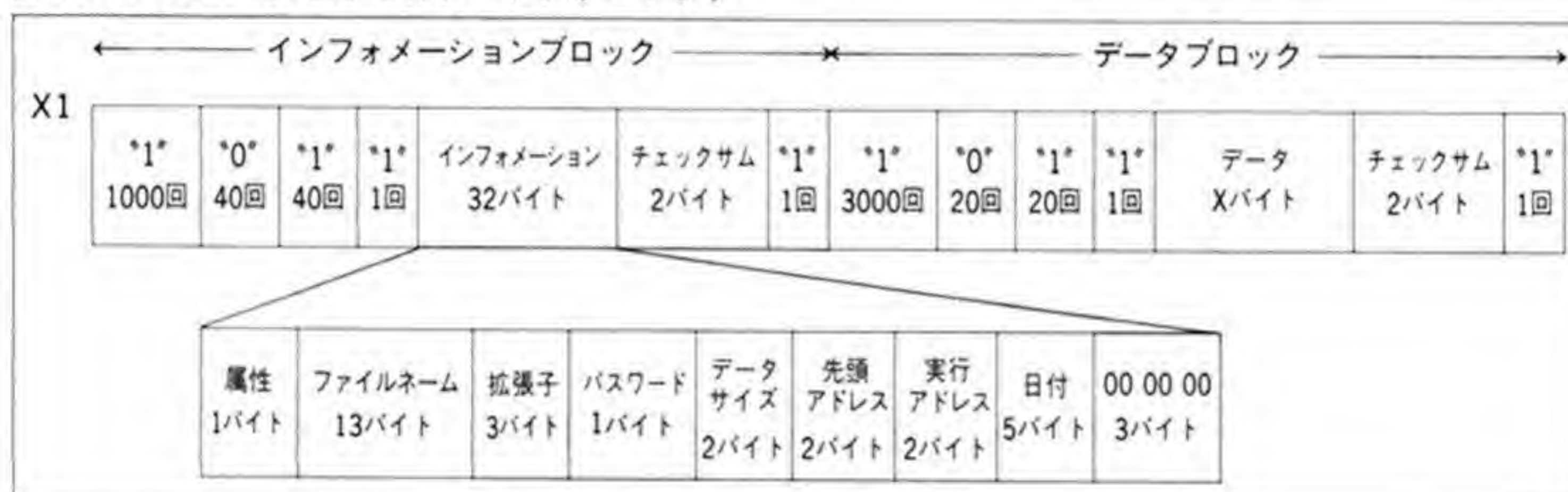
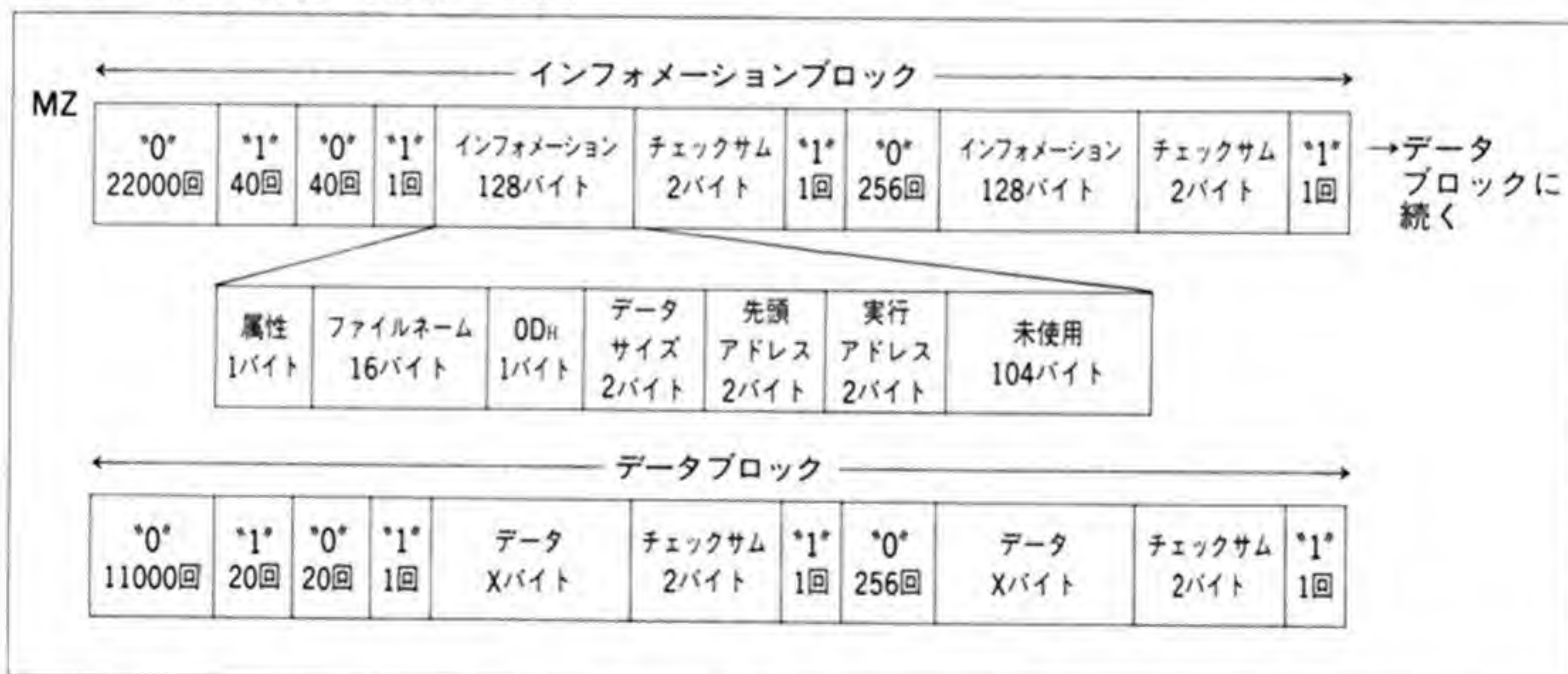


図 13-4 MZ でのテープフォーマット



図には書いてないが、実はまず最初に無録音部が 8 秒間ある。これはテープのリーダーの読み飛ばしと、APSS のために必要である。APSS, 別名「一発頭出し」は簡単に言ってしまえば「早送りしながら無録音部を検出して止まる」ということだからである。続いてインフォメーション部とデータ部が来るわけである。これらの基本的な構造は同じであるからインフォメーション部だけをねっとりと説明する。

インフォメーション部は図 13-3 に示してあるようなマークから始まる。つまり「1」が



1000 個,「0」が 40 個, また「1」が 40 個, 最後に駄目押しで「1」が 1 個である。その後が続くのがインフォメーションの本体であるが, そのバイト数は 32 バイトと決まっている。

さて, そこで「1 バイト」の書き方である。まず 1 バイトに先立って**スタートビット**なるものが書かれる。これは「1」が 1 回である。なぜこのよーなものがあるかというと, やはりこれはテープが宿命的に持つ悲しさと言わざるを得ない。つまり, テープはメカの回転変動などの影響を受けやすいので, 所々に印を入れておかなければならないのだ。ただしこれを入れるとエラー発生率が下がるというものではなく, エラーの発生を検知できるというぐらいの役割しかない。それともう一つ, 先程述べた**マーク**と同じパターンがテープに書かれるということのを避ける意味もあるだろう。9 ビットごとに「1」が来るのだから, ファイル名を書いている最中に偶然マークと同じパターン（「0」が続けて 20 個）が現れてしまう心配はないのである（こちらの方がもっと大事であろう）。そのよーな手続きを経た後に, 1 バイトを書くのである。書く順番は第 7 ビットからである。

このようにして 32 バイトのインフォメーションを書いた後に続いて, 2 バイトのチェックサムがある。これは「チェックサムエラー」を出すためである。図 13-3 にもあるように, これはインフォメーションブロックにある 32 バイトのデータ中にある「1」の個数である。この値は変態なことに, High, Low の順になっている。つまり「1」が 300 個 = 12C<sub>H</sub> 個だったなら, この 2 バイトは 01<sub>H</sub>, 2C<sub>H</sub> の順に並んでいる（インフォメーションの場合は 32 バイトだから, 最大でも 100<sub>H</sub> であるが）。Z80 での通常の 2 バイト数値とは逆順なのである。

インフォメーション部は最後に「1」が 1 回書かれて終わる。これは**ストップビット**と呼ばれる。

データブロックは, マークが少し違うこと, 長さ（バイト数）が不定であることを除けばインフォメーションブロックと同じである。データブロックのバイト数はインフォメーションブロックの 18, 19 バイト目（「0 バイト目」から数える）にあるから, このことを心得ていれば問題はない。

## 読んだり書いたりするものである

ここからプログラムの解説モードに突入する。

リスト 13-1 はテープのリード/ライトを行なう機械語プログラム, リスト 13-2 はそのダンプリスト, リスト 13-3 がそれを使っているプログラムである。ただし恐ろしいことに, 普通では満足できないというその筋の「さが」によって, さまざまな仕掛けがしてあるのだ。すなわち,

- ボーレーを自由に設定できる
- G-RAM を使って 64K バイトを読み書きできる
- フォーマットは MZ か X1 を選択できる

となっているのである。その結果として使い方は少々その筋しているが, パターンは限ら



れているのだからこれでいーのだ。

リスト 13-1 テープ入出力プログラム

		.Z80	.PHASE	0BC00H	
1A01		FMCMT	EQU	1A01H	
1A03		C8255	EQU	1A03H	
1A02		TOCMT	EQU	1A02H	
0072		PATRMM	EQU	0072H	;LD (HL),D:NOP
51ED		PATRIO	EQU	51EDH	;OUT (C),D
0056		PATWMM	EQU	0056H	;LD D,(HL):NOP
50ED		PATWIO	EQU	50EDH	;IN D,(C)
0041		X1L	EQU	65	
001D		X1S	EQU	29	
002E		X1W	EQU	46	
		;4 JUMP TABLES			
BC00	C3 BC27	WHEAD:	JP	WHEAD1	
BC03	C3 BC4A	WBODY:	JP	WBODY1	
BC06	C3 BD30	RHEAD:	JP	RHEAD1	
BC09	C3 BD47	RBODY:	JP	RBODY1	
BC0C	BE60	FCBAD:	DW	LAST	
BC0E	0000	BDYLEN:	DW	0000H	
BC10	00	FLAG:	DB	0	;MZ or X1
BC11	41	LONGH:	DB	X1L	;LONG HIGH
BC12	33	LONGL:	DB	X1L-14	;LONG LOW
BC13	1D	SHORTH:	DB	X1S	;SHORT HIGH
BC14	0F	SHORTL:	DB	X1S-14	;SHORT L
BC15	2E	W185:	DB	X1W	;SAMPLING TIME
BC16	05	AFTERW:	DB	5	;SKIP
BC17		ISSET:			;FCB MARK
BC17	03E8	IMARK1:	DW	1000	
BC19	0028	IMARK2:	DW	40	
BC1B	0028	IMARK3:	DW	40	
BC1D		DSET:			;BODY MARK
BC1D	0BB8	DMARK1:	DW	3000	
BC1F	0014	DMARK2:	DW	20	
BC21	0014	DMARK3:	DW	20	
BC23	0020	FCBLEN:	DW	32	;FCB LENGTH
BC25		SUM:	DS	2	;WORK AREA
		;***** WRITE ROUTINE BEGIN *****			
BC27	AF	WHEAD1:	XOR	A	;WRITE HEAD
BC28	01 1A03		LD	BC,C8255	;8255 CONTROLE
BC2B	ED 79		OUT	(C),A	;WRITE 'L'
BC2D	21 21E4		LD	HL,8676	;COUNTER
BC30	CD BE3F		CALL	WAITXX	;WRITE 8 SEC GAP
BC33	F3		DI		
BC34	21 0056		LD	HL,PATWMM	
BC37	22 BCBB		LD	(PATW),HL	;PATCH!
BC3A	2A BC0C		LD	HL,(FCBAD)	
BC3D	ED 5B BC23		LD	DE,(FCBLEN)	
BC41	FD 21 BC17		LD	IY,ISSET	
BC45	CD BC6A		CALL	WRITGO	;WRITE HEAD
BC48	FB		EI		
BC49	C9		RET		
BC4A	21 0100	WBODY1:	LD	HL,100H	;WRITE BODY
BC4D	CD BE3F		CALL	WAITXX	;WRITE GAP
BC50	21 50ED		LD	HL,PATWIO	
BC53	22 BCBB		LD	(PATW),HL	;PATCH!
BC56	21 FF00		LD	HL,0000H-100H	;0FF00H
BC59	01 4000		LD	BC,4000H	;GRAM
BC5C	ED 5B BC0E		LD	DE,(BDYLEN)	
BC60	FD 21 BC1D		LD	IY,DSET	



BC64	F3	DI	
BC65	CD BC6A	CALL	WRITGO ;WRITE BODY
BC68	FB	EI	
BC69	C9	RET	
;			
BC6A	D9	WRITGO: EXX	
BC6B	FD 4E 00	LD	C,(IY+0)
BC6E	FD 46 01	LD	B,(IY+1)
BC71	37	SCF	;CARRY=1
BC72	CD BCE3	CALL	WMARK ;WRITE MARK1
;			
BC75	FD 4E 02	LD	C,(IY+2)
BC78	FD 46 03	LD	B,(IY+3)
BC7B	B7	OR	A ;CARRY=0
BC7C	CD BCE3	CALL	WMARK ;WRITE MARK2
;			
BC7F	FD 4E 04	LD	C,(IY+4)
BC82	FD 46 05	LD	B,(IY+5)
BC85	37	SCF	;CARRY=1
BC86	CD BCE3	CALL	WMARK ;WRITE MARK3
;			
BC89	37	SCF	;CARRY=1
BC8A	CD BCFB	CALL	WBIT ;START BIT
;			
BC8D	D9	EXX	
BC8E	CD BC96	CALL	WBYTES ;HONTAI
;			
BC91	37	SCF	;CARRY=1
BC92	CD BCFB	CALL	WBIT ;STOP BIT
BC95	C9	RET	
;			
BC96	D9	WBYTES: EXX	
BC97	21 0000	LD	HL,0000H ;CHECK SUM
BC9A	D9	EXX	
BC9B	CD BCB5	CALL	WBS1
;			
BC9E	D9	EXX	
BC9F	7C	LD	A,H
BCA0	65	LD	H,L
BCA1	6F	LD	L,A ;SWAP H-L
BCA2	22 BC25	LD	(SUM),HL
BCA5	11 0002	LD	DE,2
BCA8	21 0056	LD	HL,PATWMM
BCAB	22 BCB8	LD	(PATW),HL ;PATCH!
BCAE	21 BC25	LD	HL,SUM
BCB1	CD BCB5	CALL	WBS1 ;WRITE CHECK SUM
BCB4	C9	RET	
;			
BCB5	D5	WBS1: PUSH	DE
BCB6	78	LD	A,B
BCB7	FE 40	CP	40H ;BC < 4000H ?
BCB9	38 05	JR	C,MRAMW ;THEN JUMP
BCBB	ED 50	PATW: IN	D,(C) ;GET FROM GRAM
BCBD	C3 BCC1	JP	MRAMW1
BCC0	56	MRAMW: LD	D,(HL) ;GET FROM MRAM
BCC1	23	MRAMW1: INC	HL
BCC2	03	INC	BC ;INC POINTERS
BCC3	CD BCCD	CALL	WBYTE ;WRITE 1 BYTE
BCC6	D1	POP	DE
BCC7	1B	DEC	DE ;DEC COUNTER
BCC8	7A	LD	A,D
BCC9	B3	OR	E ;CHECK IT
BCCA	20 E9	JR	NZ,WBS1
BCCC	C9	RET	
;			
BCCD	C5	WBYTE: PUSH	BC ;WRITE 1 BYTE
BCCE	37	SCF	
BCCF	CD BCFB	CALL	WBIT
;			
BCD2	06 08	LD	B,8 ;8 BITS
BCD4	CB 12	WBYTE1: RL	D
BCD6	D9	EXX	
BCD7	D2 BCDB	JP	NC,WBYTE2
BCDA	23	INC	HL ;INC CHECK SUM
BCDB	D9	WBYTE2: EXX	
BCDC	CD BCFB	CALL	WBIT ;WRITE 1 BIT
BCDF	10 F3	DJNZ	WBYTE1
BCE1	C1	POP	BC
BCE2	C9	RET	
;			
BCE3	F5	WMARK: PUSH	AF ;SAVE CARRY



BCE4	3A BC10	LD	A,(FLAG);GET FLAG	
BCE7	CE 00	ADC	A,00H	
BCE9	1F	RRA	;IF MZ CY REVERSE	
BCEA	F5	WMARK1: PUSH	AF	
BCEB	C5	PUSH	BC	
BCEC	CD BCFB	CALL	WBIT	
BCEF	C1	POP	BC	
BCF0	0B	DEC	BC	
BCF1	78	LD	A,B	
BCF2	B1	OR	C ;COUNTER	
BCF3	28 03	JR	Z,WMARK2	
BCF5	F1	POP	AF	
BCF6	18 F2	JR	WMARK1	
BCF8	F1	WMARK2: POP	AF	
BCF9	F1	POP	AF	
BCFA	C9	RET		
BCFB		WBIT: ;CY->"1",NON CY->"0"		
BCFB	C5	PUSH	BC ;11	
BCFC	30 09	JR	NC,WBIT0 ;12 or 7	
BCFE	3A BC12	LD	A,(LONGL) ;13	
BD01	08	EX	AF,AF' ;4	
BD02	3A BC11	LD	A,(LONGH) ;13	
BD05	18 0B	JR	WBIT1 ;12	
BD07	38 09	WBIT0: JR	C,WBIT1 ;12 or 7	
BD09	3A BC14	LD	A,(SHORTL) ;13	
BD0C	08	EX	AF,AF' ;4	
BD0D	3A BC13	LD	A,(SHORTH) ;13	
BD10	18 00	JR	WBIT1 ;12	
BD12	01 1A03	WBIT1: LD	BC,C8255 ;10	
BD15	F5	PUSH	AF ;11	
BD16	3E 01	LD	A,1 ;7	
BD18	ED 79	OUT	(C),A ;12	
BD1A	F1	POP	AF ;10	
BD1B	CD BE52	CALL	WAITX ;17	
BD1E	AF	XOR	A ;4	
BD1F	ED 79	OUT	(C),A ;12	
BD21	08	EX	AF,AF' ;4	
BD22	CD BE52	CALL	WAITX ;17	
BD25	01 1A01	LD	BC,FMCMT ;10	
BD28	ED 78	IN	A,(C) ;12	
BD2A	0F	RRCA	;4	
BD2B	D2 BE0D	JP	NC,BREAK ;10	
BD2E	C1	POP	BC ;10	
BD2F	C9	RET	;10	
		***** WRITE ROUTINE END *****		
		***** READ ROUTINE BEGIN *****		
BD30	F3	RHEAD1: DI	;READ HEAD	
BD31	21 0072	LD	HL,PATRMM	
BD34	22 BDC6	LD	(PATR),HL ;PATCH!	
BD37	2A BC0C	LD	HL,(FCBAD) ;DCB ADD.	
BD3A	ED 5B BC23	LD	DE,(FCBLEN) ;LENGTH	
BD3E	FD 21 BC17	LD	IY,ISET	
BD42	CD BD61	CALL	READGO	
BD45	FB	EI		
BD46	C9	RET		
BD47	F3	RBODY1: DI	;READ BODY	
BD48	21 51ED	LD	HL,PATRIO	
BD4B	22 BDC6	LD	(PATR),HL ;PATCH!	
BD4E	21 FF00	LD	HL,0000H-100H ;DATA ADD.	
BD51	01 4000	LD	BC,4000H	
BD54	ED 5B BC0E	LD	DE,(BDYLEN) ;LENGTH	
BD58	FD 21 BC1D	LD	IY,DSET	
BD5C	CD BD61	CALL	READGO	
BD5F	FB	EI		
BD60	C9	RET		
BD61	D9	READGO: EXX		
BD62	CD BDFD	REDGO1: CALL	COUNT0 ;COUNT "0"	
BD65	FD 5E 02	LD	E,(IY+2)	
BD68	FD 56 03	LD	D,(IY+3)	
BD6B	B7	OR	A	
BD6C	ED 52	SBC	HL,DE ;CHECK MARK2	
BD6E	20 F2	JR	NZ,REDGO1	



BD70	FD 46 04		
BD73	05		
BD74	3A BC10		
BD77	57		
BD78	CD BE14		
BD7B	3E 00		
BD7D	17		
BD7E	BA		
BD7F	CA BE0D		
BD82	10 F4		
BD84	CD BE14		
BD87	D2 BD62		
BD8A	D9		
BD8B	CD BDA2		
BD8E	ED 5B BC25		
BD92	7A		
BD93	53		
BD94	5F		
BD95	B7		
BD96	ED 52		
BD98	C2 BE11		
BD9B	CD BE14		
BD9E	D2 BE0D		
BDA1	C9		
BDA2	D9		
BDA3	21 0000		
BDA6	D9		
BDA7	CD BDBD		
BDAA	D9		
BDAB	E5		
BDAC	11 0002		
BDAF	21 0072		
BDB2	22 BDC6		
BDB5	21 BC25		
BDB8	CD BDBD		
BDBB	E1		
BDBC	C9		
BDBD	D5		
BDBE	CD BDD5		
BDC1	78		
BDC2	FE 40		
BDC4	38 05		
BDC6	ED 51		
BDC8	C3 BDCC		
BDCB	72		
BDCC	23		
BDCD	03		
BDCE	D1		
BDCF	1B		
BDD0	7A		
BDD1	B3		
BDD2	20 E9		
BDD4	C9		
BDD5	C5		
BDD6	CD BE14		
BDD9	30 32		
Bddb	16 00		
Bddd	06 08		
Bddf	CD BE14		
BDE2	D9		
BDE3	30 01		
BDE5	23		
BDE6	D9		
BDE7	CB 12		
BDE9	10 F4		
BDEB	C1		
BDEC	C9		
BDED	21 0000		
BDF0	3A BC10		

		LD	B, (IY+4)
		DEC	B
		LD	A, (FLAG)
		LD	D, A
REDGO2:	CALL	RBIT	
	LD	A, 00H	
	RLA		;X1->1, MZ->0
	CP	D	;BIT = FLAG ?
	JP	Z, BREAK	;THEN BREAK
	DJNZ	REDGO2	
	CALL	RBIT	;READ LAST MARK
	JP	NC, REDGO1	
	EXX		
	CALL	RBYTES	;READ HONTAI
	LD	DE, (SUM)	
	LD	A, D	
	LD	D, E	
	LD	E, A	;SWAP H-L
	OR	A	
	SBC	HL, DE	;CHECK SUM
	JP	NZ, CSERR	
	CALL	RBIT	;STOP BIT
	JP	NC, BREAK	
	RET		
RBYTES:	EXX		
	LD	HL, 0000H	;CHECK SUM
	EXX		
	CALL	RBS1	;READ HONTAI
	EXX		
	PUSH	HL	
	LD	DE, 2	
	LD	HL, PATRMM	
	LD	(PATR), HL	;PATCH!
	LD	HL, SUM	
	CALL	RBS1	;READ CHECK SUM
	POP	HL	
	RET		
RBS1:	PUSH	DE	
	CALL	RBYTE	
	LD	A, B	
	CP	40H	;BC < 4000H ?
	JR	C, MRAMR	;THEN JUMP
PATR:	OUT	(C), D	;STORE TO GRAM
	JP	MRAMR1	
MRAMR:	LD	(HL), D	;STORE TO MRAM
MRAMR1:	INC	HL	
	INC	BC	
	POP	DE	
	DEC	DE	
	LD	A, D	
	OR	E	;CHECK COUNTER
	JR	NZ, RBS1	
	RET		
RBYTE:	PUSH	BC	;READ 1 BYTE
	CALL	RBIT	
	JR	NC, BREAK	;START BIT
	LD	D, 00H	;CLEAR DATA
	LD	B, 8	
RBYTE1:	CALL	RBIT	
	EXX		
	JR	NC, RBYTE2	
	INC	HL	;INC CHECK SUM
RBYTE2:	EXX		
	RL	D	;STORE 1 BIT
	DJNZ	RBYTE1	
	POP	BC	
	RET		
COUNT1:	LD	HL, 0000H	
	LD	A, (FLAG)	



BDF3	B7	OR	A	
BDF4	20 10	JR	NZ,CT01 ;IF MZ COUNT0	
BDF6	CD BE14	CT11:	CALL	RBIT
BDF9	D0		RET	NC
BDFA	23		INC	HL
BDFB	18 F9		JR	CT11
;				
BDFD	21 0000	COUNT0:	LD	HL,0000H
BE00	3A BC10		LD	A,(FLAG)
BE03	B7		OR	A
BE04	20 F0		JR	NZ,CT11 ;IF MZ COUNT1
BE06	CD BE14	CT01:	CALL	RBIT
BE09	D8		RET	C
BE0A	23		INC	HL
BE0B	18 F9		JR	CT01
;				
BE0D	3E 1D	BREAK:	LD	A,29 ;ERR CODE
BE0F	DD E9		JP	(IX)
BE11	AF	CSERR:	XOR	A ;UNPRINTABLE
BE12	DD E9		JP	(IX)
;				
BE14	C5	RBIT:	PUSH	BC
BE15	01 1A01		LD	BC,FMCMT
BE18	ED 78	RBIT1:	IN	A,(C) ;IN DATA
BE1A	0F		RRCA	
BE1B	30 F0		JR	NC,BREAK
BE1D	0F		RRCA	
BE1E	38 F8		JR	C,RBIT1 ;WHILE(HIGH) DO
;				
BE20	ED 78	RBIT2:	IN	A,(C) ;12
BE22	0F		RRCA	;4
BE23	D2 BE0D		JP	NC,BREAK ;10
BE26	0F		RRCA	;4
BE27	D2 BE20		JP	NC,RBIT2 ;10
;WHILE(LOW) DO				
;				
;NOW I GOT 'EDGE' !				
BE2A	3A BC15		LD	A,(W185) ;7
BE2D	CD BE52		CALL	WAITX ;17
BE30	ED 78		IN	A,(C)
BE32	0F		RRCA	
BE33	D2 BE0D		JP	NC,BREAK
BE36	0F		RRCA	
BE37	3A BC16		LD	A,(AFTERW) ;SKIP TIME
BE3A	DC BE52		CALL	C,WAITX
BE3D	C1		POP	BC
BE3E	C9		RET	
;				
;***** READ ROUTINE END *****				
;				
;***** WAIT ROUTINE *****				
;				
BE3F	01 1A01	WAITXX:	LD	BC,FMCMT;10
BE42	ED 78	WAPSS:	IN	A,(C) ;12
BE44	0F		RRCA	;4
BE45	D2 BE0D		JP	NC,BREAK;10
BE48	AF		XOR	A ;4
BE49	CD BE52		CALL	WAITX ;17+14*256+31
BE4C	2B		DEC	HL ;6
BE4D	7C		LD	A,H ;4
BE4E	B5		OR	L ;4
BE4F	20 F1		JR	NZ,WAPSS ;12 or 7
BE51	C9		RET	;10
;DAITAI HL*3688				
BE52	F5	WAITX:	PUSH	AF ;11
BE53	3D	WAITXL:	DEC	A ;4
BE54	C2 BE53		JP	NZ,WAITXL ;10
;				
BE57	F1		POP	AF ;10
BE58	C9		RET	;10
;11+14*Acc+20				
;Acc=46->644+31=675				
;				
MACRO-80 3.44 09-Dec-81 PAGE 1-7				
BE59		LAST:	DS	7
BE60			DS	0
END				



リスト 13-2 テープ入出力ダンプリスト

BC00	C3	27	BC	C3	4A	BC	C3	30	:	62	BD40	17	BC	CD	61	BD	FB	C9	F3	:	75
BC08	BD	C3	47	BD	60	BE	00	00	:	A2	BD48	21	ED	51	22	C6	BD	21	00	:	25
BC10	00	41	33	1D	0F	2E	05	E8	:	BB	BD50	FF	01	00	40	ED	5B	0E	BC	:	52
BC18	03	28	00	28	00	B8	0B	14	:	2A	BD58	FD	21	1D	BC	CD	61	BD	FB	:	DD
BC20	00	14	00	20	00	1A	1A	AF	:	17	BD60	C9	D9	CD	FD	BD	FD	5E	02	:	86
BC28	01	03	1A	ED	79	21	E4	21	:	AA	BD68	FD	56	03	B7	ED	52	20	F2	:	5E
BC30	CD	3F	BE	F3	21	56	00	22	:	56	BD70	FD	46	04	05	3A	10	BC	57	:	A9
BC38	BB	BC	2A	0C	BC	ED	5B	23	:	D4	BD78	CD	14	BE	3E	00	17	BA	CA	:	78
BC40	BC	FD	21	17	BC	CD	6A	BC	:	A0	-----										
BC48	FB	C9	21	00	01	CD	3F	BE	:	B0	SUM:	F3	6C	48	92	61	30	30	CF	D3FE	
BC50	21	ED	50	22	BB	BC	21	00	:	18	BD80	0D	BE	10	F4	CD	14	BE	D2	:	40
BC58	FF	01	00	40	ED	5B	0E	BC	:	52	BD88	62	BD	D9	CD	A2	BD	ED	5B	:	6C
BC60	FD	21	1D	BC	F3	CD	6A	BC	:	DD	BD90	25	BC	7A	53	5F	B7	ED	52	:	03
BC68	FB	C9	D9	FD	4E	00	FD	46	:	2B	BD98	C2	11	BE	CD	14	BE	D2	0D	:	0F
BC70	01	37	CD	E3	BC	FD	4E	02	:	F1	BDA0	BE	C9	D9	21	00	00	D9	CD	:	27
BC78	FD	46	03	B7	CD	E3	BC	FD	:	66	BDA8	BD	BD	D9	E5	11	02	00	21	:	6C
-----											BDB0	72	00	22	C6	BD	21	25	BC	:	19
SUM:	D9	80	90	9D	3E	3C	75	78	C262		BDB8	CD	BD	BD	E1	C9	D5	CD	D5	:	68
BC80	4E	04	FD	46	05	37	CD	E3	:	81	BDC0	BD	78	FE	40	38	05	ED	51	:	EE
BC88	BC	37	CD	FB	BC	D9	CD	96	:	B3	BDC8	C3	CC	BD	72	23	03	D1	1B	:	D0
BC90	BC	37	CD	FB	BC	C9	D9	21	:	3A	BDD0	7A	B3	20	E9	C9	C5	CD	14	:	A5
BC98	00	00	D9	CD	B5	BC	D9	7C	:	6C	BDD8	BE	30	32	16	00	06	08	CD	:	11
BCA0	65	6F	22	25	BC	11	02	00	:	EA	BDE0	14	BE	D9	30	01	23	D9	CB	:	A3
BCA8	21	56	00	22	BB	BC	21	25	:	56	BDE8	12	10	F4	C1	C9	21	00	00	:	C1
BCB0	BC	CD	B5	BC	C9	D5	78	FE	:	0E	BDF0	3A	10	BC	B7	20	10	CD	14	:	CE
BCB8	40	38	05	ED	50	C3	C1	BC	:	FA	BDF8	BE	D0	23	18	F9	21	00	00	:	E3
BCC0	56	23	03	CD	CD	BC	D1	1B	:	BE	-----										
BCC8	7A	B3	20	E9	C9	C5	37	CD	:	C8	SUM:	E6	60	6B	FF	80	86	6E	37	795A	
BCD0	FB	BC	06	08	CB	12	D9	D2	:	4D	BE00	3A	10	BC	B7	20	F0	CD	14	:	AE
BCD8	DB	BC	23	D9	CD	FB	BC	10	:	27	BE08	BE	D8	23	18	F9	3E	1D	DD	:	02
BCE0	F3	C1	C9	F5	3A	10	BC	CE	:	46	BE10	E9	AF	DD	E9	C5	01	01	1A	:	3F
BCE8	00	1F	F5	C5	CD	FB	BC	C1	:	1E	BE18	ED	78	0F	30	F0	0F	38	F8	:	D3
BCF0	0B	78	B1	28	03	F1	18	F2	:	5A	BE20	ED	78	0F	D2	0D	BE	0F	D2	:	F2
BCF8	F1	F1	C9	C5	30	09	3A	12	:	F5	BE28	20	BE	3A	15	BC	CD	52	BE	:	C6
-----											BE30	ED	78	0F	D2	0D	BE	0F	3A	:	5A
SUM:	DD	D3	D0	37	2A	8D	0F	52	7B60		BE38	16	BC	DC	52	BE	C1	C9	01	:	49
BD00	BC	08	3A	11	BC	18	0B	38	:	26	BE40	01	1A	ED	78	0F	D2	0D	BE	:	2C
BD08	09	3A	14	BC	08	3A	13	BC	:	24	BE48	AF	CD	52	BE	2B	7C	B5	20	:	08
BD10	18	00	01	03	1A	F5	3E	01	:	6A	BE50	F1	C9	F5	3D	C2	53	BE	F1	:	B0
BD18	ED	79	F1	CD	52	BE	AF	ED	:	D0	BE58	C9	00	00	00	00	00	00	00	:	C9
BD20	79	08	CD	52	BE	01	01	1A	:	7A	-----										
BD28	ED	78	0F	D2	0D	BE	C1	C9	:	9B	SUM:	48	29	33	66	5E	E9	DC	9D	A9B4	
BD30	F3	21	72	00	22	C6	BD	2A	:	55											
BD38	0C	BC	ED	5B	23	BC	FD	21	:	0D											

リスト 13-3 BASIC から「リスト 13-2」を試してみる

```

100 CLEAR &HBC00
110 DEFUSR0=&HBC00:'WRITE FCB
120 DEFUSR1=&HBC03:'WRITE BODY
130 DEFUSR2=&HBC06:'READ FCB
140 DEFUSR3=&HBC09:'READ BODY
150 '
160 INIT
170 IF MEM$(&HBC00,3)<>HEXCHR$("C3 27 BC")THEN LOADM "TAPE.OBJ"
180 '
190 'for X1:ハイロノ 2700 本-
200 MEM$(&HBC10,16)=HEXCHR$("00 41 33 1D 0F 2E 05 E8 03 28 00 28 00 B8 0B 14")
210 MEM$(&HBC20,5)=HEXCHR$("00 14 00 20 00")
220 '
230 'for S-OS "T":2400 本- ナノニ X1 ト オナシ 本-レート タ ッタリ スル
240 MEM$(&HBC10,16)=HEXCHR$("01 41 33 1D 0F 2E 01 F0 55 28 00 28 00 F8 2A 14")
250 MEM$(&HBC20,5)=HEXCHR$("00 14 00 80 00")
260 '
270 'for MZ-80B/2000,2200:2000 本-
280 MEM$(&HBC10,16)=HEXCHR$("01 58 56 28 26 40 05 F0 55 28 00 28 00 F8 2A 14")

```



```

290 'MEM$(&HBC20,5) =HEXCHR$("00 14 00 80 00")
300 '
310 'for MZ-80K/C/1200/700/1500:1200 * -
320 'MEM$(&HBC10,16)=HEXCHR$("01 83 81 3E 3C 5A 05 F0 55 28 00 28 00 F8 2A 14")
330 'MEM$(&HBC20,5) =HEXCHR$("00 14 00 80 00")
340 '
350 INPUT "R/W";A$:IF A$="R" THEN GOSUB"READ":GOTO 350
360 IF A$="W" THEN GOSUB"WRITE":GOTO 350
370 GOTO 350
380 '
390 LABEL"READ"
400 CMT=2:'PLAY
410 D$=USR2(""): 'READ FCB
420 CMT=1:'CSTOP
430 PRINT #0 "FOUND",MEM$(&HBE61,16),
440 L=CVI(MEM$(&HBE60+18,2)):'LENGTH
450 S=CVI(MEM$(&HBE60+20,2)):'START ADDRESS
460 X=CVI(MEM$(&HBE60+22,2)):'JUMP ADDRESS
470 PRINT #0 RIGHT$("000"+HEX$(S),4),
480 PRINT #0 RIGHT$("000"+HEX$(S+L-1),4),
490 PRINT #0 RIGHT$("000"+HEX$(X),4)
500 '
510 MEM$(&HBC0E,2)=MEM$(&HBE60+18,2):'COPY LENGTH
520 CMT=2:'PLAY
530 PRINT #0 "LOADING",MEM$(&HBE61,16)
540 D$=USR3(""): 'READ BODY
550 CMT=1:'CSTOP
560 RETURN
570 '
580 LABEL"WRITE"
590 PRINT #0 "WRITING",MEM$(&HBE61,16)
600 CMT=10:'WRITE
610 D$=USR0(""): 'WRITE FCB
620 CMT=1:'CSTOP
630 '
640 MEM$(&HBC0E,2)=MEM$(&HBE60+18,2):'COPY LENGTH
650 CMT=10:'PLAY
660 D$=USR1(""): 'WRITE BODY
670 CMT=1:'CSTOP
680 RETURN

```

では機械語プログラムは後にして、先にリスト 13-3 の使い方を説明する。

よくある展開として、各モードの指定は「MEM\$」で機械語ルーチンにパラメータを渡すことで行なっている。一番最初に指定してあるのは X1 フォーマットの場合のパラメータである。大抵の用はこれで済むだろうが、それだけでは面白くないので、後続くようにさまざまなボーレートとフォーマットの挑戦を受けているのである。使うときは REM (') を適当に入れたり消したりしていただきたい。

パラメータの領域は BC10<sub>H</sub>～BC24<sub>H</sub>になっている。それぞれの意味は表 13-1 に示してある。この部分はリスト 13-1 と併せて見た方が分かりやすいだろう。

それはともかく、このリスト 13-3 は 30K バイト以上の長さのファイル (実は BASIC) を読み書きさせて試験してあるので、異機種フォーマットも大丈夫のはずである。ただし 1200 bps だけは例外で、CZ-8RL1 を使った場合は、**背面にあるボーレート切り換えスイッチを Low にしておけば OK** だが、それ以外の内蔵デッキでは、かなりエラーが出てしまうようである。ちなみに私は、しばらくの間このボーレート切り換えスイッチに気が付かず、バグ地獄をはいずり回っていたのであった。

それはさておき、具体的な説明をしておく。110～140 行に DEFUSR 文が 4 個並んでいる。機能は注釈にあるとおりである。まずは 390 行からのテープリードであるが、最初に CMT 命令でカセットデッキを READ 状態にしている。プログラム作成中は機械語ルーチンの中にカセットデッキコントロールのためのサブルーチンも入れてあったのだが、ちょ



表 13-1 リスト 1 のパラメータの説明

BC10<sub>H</sub>=フラグである。0ならX1フォーマット、1ならMZフォーマット。  
 BC11<sub>H</sub>=\*1\*のHighの長さ。ウェイトルーチンに渡すパラメータである。  
 BC12<sub>H</sub>=\*1\*のLowの長さ。  
 BC13<sub>H</sub>=\*0\*のHighの長さ。  
 BC14<sub>H</sub>=\*0\*のLowの長さ。  
 BC15<sub>H</sub>=信号の立ち上がりからサンプリングまでの待ち時間 (X1では185 $\mu$ s)。  
 BC16<sub>H</sub>=\*1\*を読んだ後のスキップ時間 (あんまりたいした意味はない)。  
 BC17, 8<sub>H</sub>=インフォメーションブロック (FCB) の最初のマークの数。  
 BC19, A<sub>H</sub>=インフォメーションブロックの2番目のマークの数。  
 BC1B, C<sub>H</sub>=インフォメーションブロックの3番目のマークの数。  
 BC1D, E<sub>H</sub>=データブロックの最初のマークの数。  
 BC1F, 20<sub>H</sub>=データブロックの2番目のマークの数。  
 BC21, 2<sub>H</sub>=データブロックの3番目のマークの数。  
 BC23, 4<sub>H</sub>=インフォメーションブロックの長さ。

っと長いし、第4章でやったサブCPUの使い方そのものなので、取り除いてしまったのである。次に来るのは「USR2 (〃)」で、これはインフォメーションブロックの読み込みである。読み込む位置(アドレス)はBC0C<sub>H</sub>からの2バイトに入っていることになっている。何もいじらなければここにはBE60<sub>H</sub>が入っているので、モニタでダンプすれば見ることができるのである。読み込み終わったなら、一旦テープを止めて(CMT=1)ファイル名を表示する。時々ファイル名の中に変なコントロールコードが交じっていることがあるので、「PRINT #0」を使っている。ここではさらに縁起ものなのでスタートアドレス、エンドアドレス、実行アドレスも表示している。

次に「USR3 (〃)」でファイルの本体を読み込むわけであるが、ここで先程言っておいた長さが出てくる。長さはBC0E<sub>H</sub>からの2バイトで指定するのだ。サンプルプログラムでは常識どおりにインフォメーションブロックの18, 19バイト目を使っている。

ファイルの本体は無条件にG-RAMの4000<sub>H</sub>(青画面)からロードされるので、ボーレートが目で見えるのである。48Kバイト(C000<sub>H</sub>バイト)を超える場合は、メインRAMのBF00<sub>H</sub>~FEFF<sub>H</sub>にロードされる。なぜそのように中途半端なアドレスなのかというと、X1のHuBASICではFF00<sub>H</sub>~を勝手に使えないからなのである。このことを一番手軽に確認するには、モニタに入ってダンプコマンドを実行してみればよい。しっかりキー入力バッファに使われてしまっているのである。てなわけで、使用に当たってはご注意ください。

次は580行からのファイルの書き込みであるが、これは似たようなものであるから割愛してしまって、さっさとリスト13-1の説明に行ってしまう。

まずは「PATRMM」というやつを見ていただきたい。察しのいい人なら分かるように、これはパッチ用のデータである。すなわち、このプログラムは自分自身を書き換えるのである。嫌いなテクニックを使ってしまって最高に気分が悪い私である。なんでこーなってしまったのかとゆーと、インフォメーションブロックやチェックサムはメインRAMへリード(もしくはメインRAMからライト)しなければならないが、データブロッ



クは無条件に G-RAM を対象にしているからである。フラグを使って分岐するとタイミングがその筋する可能性が高いのだ。ソフトウェアでタイミングを取っているプログラムの泣き所である。～MM というのがメイン RAM 用のパッチ、～IO というのが G-RAM 用のパッチである。

プログラムの実体は四つのジャンプから始まる。その次にパラメータエリアが並ぶ。では本筋の BC27<sub>H</sub>の「WHEAD1」を説明する。最初にやっているのは APSS 用の 8 秒間のブランクの書き込みである。8255 のビットセット/リセットコマンドを使っていることに注意。詳しくは第 0 章の 8255 ②の所を見ていただきたい。「WAITXX」というサブルーチンは、前述の「BREAK 信号」のチェックも行なうウエイトルーチンである。それが済んだ後にインフォメーションブロックを書き始めることになる。ソフトウェアでタイミングを取るプログラムであるから BC33<sub>H</sub>番地の「DI」で割り込みを禁止していることに注目。次に、インフォメーションブロック用のデータはメイン RAM 上にあるので、BC34<sub>H</sub>～BC39<sub>H</sub>でパッチを行なっている。これによって「PATW」(BCBB<sub>H</sub>)からの 2 バイトが、

LD (HL), D

NOP

に書き換わる。ここまできたなら、あとは HL レジスタにインフォメーションブロックのデータのアドレスを入れ (G-RAM の内容を書く場合は BC レジスタ)、DE レジスタにその長さを入れ、IY レジスタにマークの長さ (3 種類分) へのポインタを入れて「WRITGO」を CALL するわけである。

BC4A<sub>H</sub>番地からの「WBODY1」も基本的には同じである。

最初にやっているのは GAP の書き込みである。インフォメーションブロックの直後にデータを書き込むと、タコなリードルーチンが読み飛ばしてしまうかもしれないので入れてある。本式にはどーするのか知らないが、世の中は丸く収まっているよーである。その後は G-RAM 方面へのパッチを行なって、各レジスタに値を放り込んで、「WRITGO」を CALL しているだけである。

以上のように書くと実に簡単そうに思えるかもしれないが、ここから先が七面倒なのである。

BC6A<sub>H</sub>からの「WRITGO」を見ていただきたい。3 回「WMARK」を呼んでマークを書いている。このサブルーチンはキャリーフラグがセットされているなら「1」を、リセットされているのなら「0」を、BC レジスタの数だけ書くのである。ただしこれは X1 フォーマットの場合である。例の BC10<sub>H</sub>番地のフラグが 1 (MZ モード)のときは「0」、「1」が逆になる。BCE4<sub>H</sub>番地あたりを見て納得していただきたい。その後はフォーマットどおりにスタートビットを書き、本体のバイト列を書き (BC8E<sub>H</sub>の「CALL WBYTES」)、最後にストップビットを書いている。

そこで問題点は BC96<sub>H</sub>番地の「WBYTES」に移る。まず裏レジスタの HL を 0000<sub>H</sub>にしている。これはチェックサム用のカウンタである。その後下働きの「WBS1」を呼び出している。これがバイト列ライトの直接の実行者である。このルーチンから帰ってきた後はまた「WBS1」を呼び出して 2 バイトのチェックサムを書いて終わりである。



「WBS1」(BCB5<sub>H</sub>)はスタートビットとして「1」を1回書いた後(BCCE<sub>H</sub>～BCD1<sub>H</sub>)、データを1バイトずつDレジスタに取り出して「WBYTE」を使って第7ビットから順に書いている(各バイトごとに各スタートビットも入れれば、1バイトごとに9ビット書いているわけだ)。「WBYTE」はチェックサム用に、BCDA<sub>H</sub>番地で「1」を数えている。

ここでやっと「WBIT」(BCFB<sub>H</sub>)の説明に達した。このルーチンはキャリーフラグがセットされていれば「1」を、リセットされていれば「0」を書くわけである。BCFC<sub>H</sub>からBD11<sub>H</sub>までの間でその判定をして、A、A'レジスタにウェイト用の値を入れている。BD10<sub>H</sub>の無駄な「JR」は、「1」と「0」での時間差をなくするためのもので縁起ものなので入れている。念のために言っておくが、「1」は別名を「LONG」、「0」は別名を「SHORT」とも言うのである。つまり波形の長さからきた呼び方である。

「1」、「0」の書き方は少し考えれば分かるように、たとえばX1で「1」を書く場合は、I/Oの1A02<sub>H</sub>番地の第0ビットをセット(High)して250 $\mu$ sの間待ち、その後そのビットをリセット(Low)して250 $\mu$ s待つのである(プログラムでは8255のビットセット/リセットコマンドを使っている)。

実を言うと、X1の場合はリセット(Low)しておく時間は少々短くしてある。理由は、このサブルーチンからリターンして次の1ビットを書き始めるまでの間もLowを書き続けているから、このサブルーチンの中でまるまる250 $\mu$ s待っていると、Lowの期間が長くなりすぎてしまうからなのだ。この点に気を付けないと、X1ではリードエラーがバシバシ起きてしまうようである。しかしMZの方はこの点に関してはのんびりしているらしく、Lowが長すぎても問題はないようである。

BD25<sub>H</sub>～BD2D<sub>H</sub>は「BREAK 信号」のセンスである。この分の時間もLowの長さに入れてやらなければならないので、(LONGL)と(SHORTL)の値はその分小さくなっている。あ～めんどくさ。

以上でテープライトルーチンの説明を終わるが、なぜかMZフォーマットは2回セーブのはずなのにそうになっていない。これを手抜きというのである。それはともかく、ここまで読んで気が付くように、このプログラムはあんまり面白くない(作った本人が言っているのだから本当である)。ほとんど唯一の特長は、わりかしきっちりしたトップダウンになっていることぐらいである。この事態を招いたもとは、テープのリード/ライトルーチンというものは、決まりきったことを決まりきったようにやるものだからということであろう。「可能な限りに短く作る」などの目標があればまた別なのだろうが。

と、グチをこぼしながらリードルーチンの説明に入る。BD30<sub>H</sub>からである。「RHEAD1」、「RBODY1」の構造は基本的にライトルーチンと同じである。最初にマークを検出するのであるが、ここでは1番目のマークははなから無視している。理由は、「2、3番目のマークだけで充分だから」と「MZではテープのリーダー部分のせいで、1番目のマークは決まった個数にならないから」である。繰り返すが、世の中は丸く収まっているのである。

というわけで、2番、3番目のマークを目標にして同期を取って実際のデータが書かれ



ている所に迫っていくのが BD62<sub>H</sub>～BD89<sub>H</sub>である。ここまできたなら、「RBYTES」を呼び出して、指定したバイト数を読んでしまう。その後はチェックサムを確かめて、合っていれば OK、さもなければエラーである。

「RBYTES」,「RBS1」,「RBYTE」に関しては別に説明する必要はないだろう、ということで BE14<sub>H</sub>番地からの「RBIT」である。

まず、BE18<sub>H</sub>～BE1F<sub>H</sub>のループである。ここは「テープから読んだ信号が High である限りループせよ」=「Low になったら出ていいよ」ということである (BREAK 信号のチェックは別だよ)。ここを抜けてきたならば、そのときテープの信号は Low になっている。次にある BE20<sub>H</sub>～BE29<sub>H</sub>のループは「テープから読んだ信号が Low である限りループせよ」=「High になったら出ていいよ」である。これら二つのループのやっていることは、

### エッジの検出

なわけである。すなわち、テープの信号が Low → High になった直後には BE2A<sub>H</sub>番地に来ているのである。そこで、(X1 の場合は) 185 μs 待つてからもう一度テープの信号を読み取り、「1」なのか「0」なのかを見分けるのである。これがテープリードである。

プログラムでは BE37<sub>H</sub>番地から、

```
LD    A, (AFTERW)
```

```
CALL C, WAITX
```

などというものがある。これは、「1」を読んだ後は、(X1 の場合なら) 次の 1 ビットまで 500-185=315 μs も時間があり、へたをするとよけいなノイズを読んでしまうかもしれないので厄除けのつもりで入れたのだが、はっきり言ってほとんど役に立っていないよである。6 バイト損こいた。なお、ライトルーチンと同じように、MZ フォーマットを読むときも二度読みはしていない。

そのよーなわけでプログラムの説明は一段落するのであるが、ここで一般教養としてソフトウェアでタイミングを取る場合に必要な「時間の計算方法」を示しておく。

てきとーな機械語の参考書を見ると、各命令ごとに「T (ステート数)=××」という数字が書いてあるはずである。これが、「その命令を実行するのに何クロック必要か」ということである。たとえば、「LD A, n」ならば T=7 であるから、7 クロックかかるのである。そいでもって、X1 のクロックは 4MHz=4000000Hz だから、1 クロックの時間は、

$$1 \div 4000000 \text{ (Hz)} = 0.00000025 \text{ (秒)}$$

となる。すなわち 1 クロック=250 ns=0.25 μs である。このことから、「LD A, n」の実行時間は 0.25(μs)×7=1.75(μs)と計算できる。実用的には、**すべてのステート数を足し合わせた数字を 4 で割り、それに「μs」を付ければよい。**

念のために書いておくと、

$$1\text{ms (ミリ秒)} = 0.001 \text{ 秒}$$

$$1\mu\text{s (マイクロ秒)} = 0.000001 \text{ 秒}$$

$$1\text{ns (ナノ秒)} = 0.000000001 \text{ 秒}$$

である。



そのよーなわけでテープであった。本章のサンプルプログラムは、このままでは単なる BASIC テープをバックアップするぐらいのことしかできない。しかし頑張れば「テープ→フロッピーディスク転送プログラム」ぐらいにはなるだろう。また、G-RAM にロードされたプログラムをメイン RAM に転送して実行する機械語ルーチンなどもおいしそうである。

最後に言っておくが、このプログラムは**プロテクトのかかったゲームプログラムなどは読めない**。読めないったら読めない。まかり間違っても「某ゲームのテープ版を読めないんですけど、バグじゃないですか」などという文句をつけてこないよーに。ボーレートを自由に変えられる所まではやったのだから、その先の「不毛なあの筋」は自由研究である。





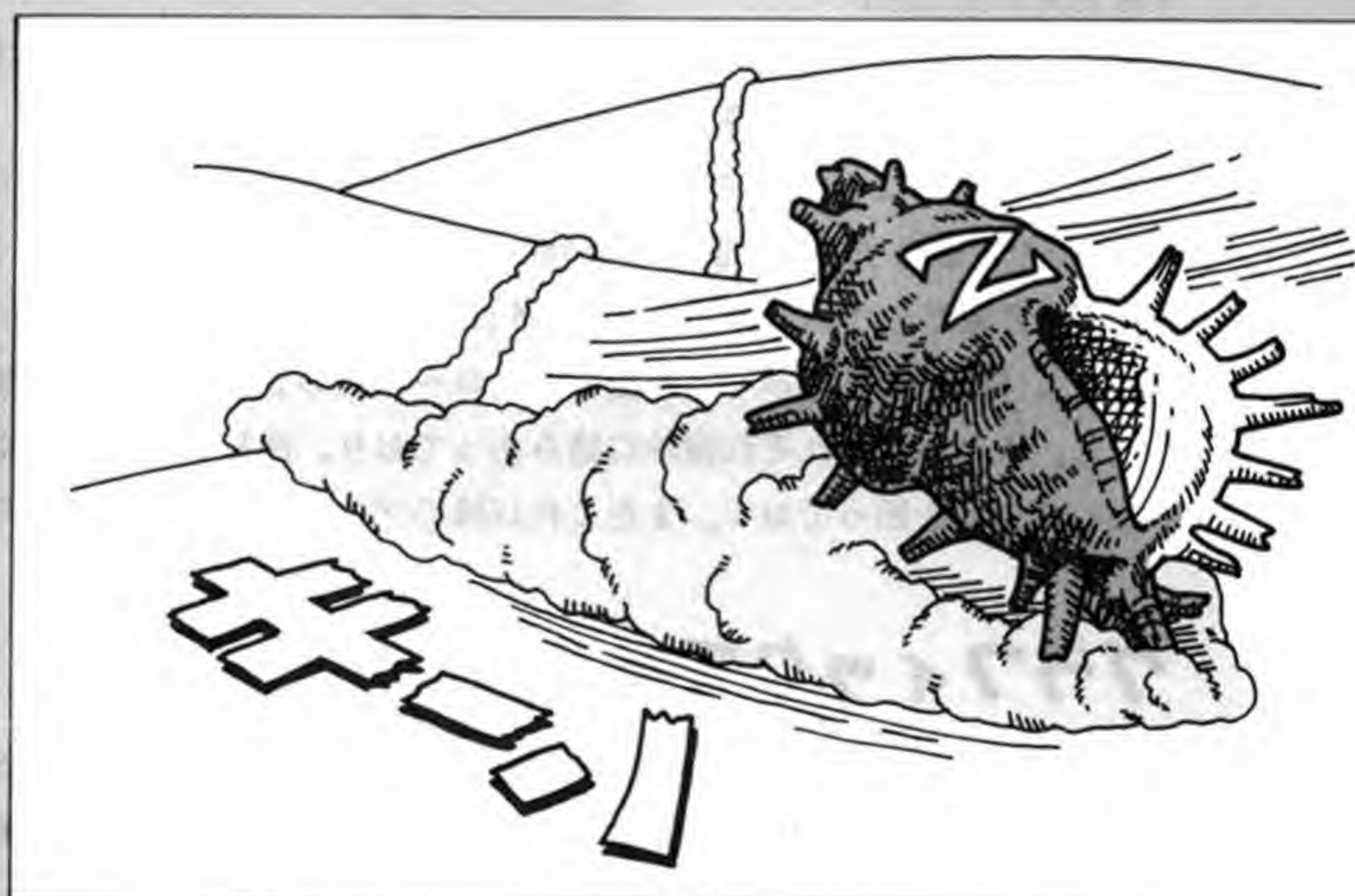


第

14

章

# turbo Z



Zの機能はおいしいのである



## ■ 第14章

# Zの機能はおいしいのである……

この章では turboZ の機能についてやるのである。さっさと書いてしまうと、turbo (turboII も含む) と turboZ の差は、

- 1) アナログ RGB にも対応し 4096 色を使用可能
- 2) 各種モード付きの画像取り込み機能
- 3) 第2水準 ROM を標準搭載
- 4) 2HD ドライブ搭載
- 5) グラフィック 2 画面をプライオリティ付きで重ね合わせ表示可能
- 6) テキストパレットを使用可能
- 7) FM 音源を標準で搭載
- 8) マウスを標準装備
- 9) スクロールイン/アウト機能
- 10) インタレーススーパーインポーズ
- 11) クロマキー (画面合成) 可能

などとなっている。

なお、turbo と turboZ の間には turboIII という微妙に緊張した関係のマシンもあるが、それについては自由研究としておく。

さて、以上のことをつらつら考えるに、turboZ にあって turbo にはない機能というのは、ただただひたすらに画面周りだけなのである。付けようと思えば 2HD ドライブだろうが、FM 音源だろうが、第2水準 ROM だろうが付いてしまうのである。そこで、「グラフィック」、「画像取り込み」、「画面コントロール」の3点に絞って取りあげることにした。ただし、これらはそれぞれ微妙に絡みあっており、決してバラバラに独立したものではないということを断っておく。またこれ以降のプログラムは CZ-8FB02 を使うこと。

## グラフィックである

CZ-8FB03 を使えばちょいちょいと 4096 色が使えののだが、昔ながらの CZ-8FB02 だとなかなかそうはいかない。そこでリスト 14-1 である。これは 4096 色の色見本である。グラフィック画面は 320×200 であるから、全部で 64000 ドットなわけだ。それを 4096 の色数で公平に配給すると、1 色当たり 15.625 ということになる。よってあれこれ考えて、1 色当たり縦 3 ドット×横 4 ドットの 12 ドットを、さらに縦横 64 色ずつ並べることにした。その処理をしているのが 250~400 行である。カラーコードの並び方は、画面左上から右にかけて順に 0, 1, 2, 3, 4, …… , 63 となり、一段下がって左から 64, 65, 66…… という、ごく当たり前の配列になっている。で、おそらくは読者の中にはどー



してこのボックスフル (LINE~BF) の結果が 4096 色の色見本を作るのか分からない人がいるかもしれない。ま、それはそれでいいのである。「どしてかなー？」と考え続けていれば、そのうち分かるときがくれば分かるであろう。もしも分からなかったとしても、それはそれで別に悩むこともない。

リスト 14-1 4096 色見本+Z 用パレットルーチン

```

100 CLEAR &HEF00
110 MEM$(&HEF00,16)=HEXCHR$("DD E5 FE 03 3E 0D C0 78 FE 04 3E 05 D8 E1 01 C5")
120 MEM$(&HEF10,16)=HEXCHR$("1F C5 3E 80 ED 79 EB 7E 23 ED 67 0F 0F 0F 4F")
130 MEM$(&HEF20,16)=HEXCHR$("7E 23 E6 F0 57 ED 67 06 10 ED 79 04 7E 23 B2 ED")
140 MEM$(&HEF30,16)=HEXCHR$("79 04 7E E6 0F B2 ED 79 C1 AF ED 79 C9 01 C5 1F")
150 MEM$(&HEF40,16)=HEXCHR$("C5 3E 80 ED 79 CD 4D EF C1 AF ED 79 C9 11 10 10")
160 MEM$(&HEF50,16)=HEXCHR$("AF 67 6F 4F AF 42 ED 79 04 ED 61 04 ED 69 19 C6")
170 MEM$(&HEF60,16)=HEXCHR$("11 30 F2 0C C8 AC 20 EC 67 2C 18 E9 00 00 00 00")
180 DEFUSR0=&HEF00
190 '
200 INIT:KLIST 0
210 OPTION SCREEN 0:WIDTH 40,25,0,1
220 OUT &H1FB0,&H80
230 GOSUB"CLS12"
240 'CALL &HEF3D 'XPINIT
250 FOR L=0 TO 5
260   D=4*(2^L)
270   IF L<4 THEN C=1 ELSE C=2
280   S=3-(L MOD 4):SCREEN S,S
290   FOR J=0 TO 255 STEP D*2
300     LINE(J+D,0)-(J+2*D-1,191),XOR,C,BF
310   NEXT
320 NEXT
330 FOR L=6 TO 11
340   D=3*(2^(L-6))
350   IF L<8 THEN C=2 ELSE C=4
360   S=3-(L MOD 4):SCREEN S,S
370   FOR J=0 TO 191 STEP D*2
380     LINE(0,J+D)-(255,J+2*D-1),XOR,C,BF
390   NEXT
400 NEXT
410 FOR I=0 TO 4095
420   FOR C1=0 TO 4095
430     C2=C1+1
440     'GOSUB"XPALET"
450     AS=USR0(MKI$(C1)+MKI$(C2))
460   NEXT
470 NEXT
480 END
490 '
500 LABEL"CLS12"
510 SCREEN,0:CLS4
520 SCREEN,1:CLS4
530 SCREEN,2:CLS4
540 SCREEN,3:CLS4
550 INIT
560 RETURN
570 '
580 LABEL"XPALET"
590 OUT &H1FC5,&H80
600 GR=INT((C1 AND &HFFF)/&H10)
610 B=(C1 AND &HF)*16
620 B0=INT(C2 AND &HF)
630 R0=INT((C2 AND &HF0)/&H10)
640 G0=INT((C2 AND &HF00)/&H100)
650 OUT &H1000+GR,B+B0
660 OUT &H1100+GR,B+R0
670 OUT &H1200+GR,B+G0
680 OUT &H1FC5,&H0
690 RETURN

```

で、色見本だけではナニなので、410~470 行でパレットをいじっている。見れば分かると思うが、最初のループ (0 回目) で「PALET △, △」=初期化、次の 1 回目のループで「PALET △, △+1」、さらに「PALET △, △+2」……をやっているわけだ。これは無限ループになっているので止めるには **BREAK** キーである。450 行の USR0 を取り去



って440行のGOSUB文を生かすと、パレット変更ルーチンが580行からのBASIC部分になって、とても遅くなるので一度(一度で充分)試していただきたい。USR0で実行している機械語サブルーチンのアセンブルリストはリスト14-2である。で、それをよく見るとなんか変なのが後ろにくっついていてるわけだ。そう、これはパレット初期化ルーチンなわけだ。リスト14-1の240行にこっそりとあるように、「CALL &HEF3D」で4096色のパレットを初期化する。EF3D<sub>H</sub>~EF6B<sub>H</sub>は簡単にリロケータブルになるので(自分でやっていただきたい)、結構便利だろうと思う。ちなみにこのルーチンのメイン部分であるEF4D<sub>H</sub>~EF6B<sub>H</sub>(計31バイト)はその筋にコンパクト設計となっている。かなり苦勞して作ったからこれよりも小さく作るとはかなり難しいであろう。それよりも、きっと解説するだけでも一苦勞だと思う。ちなみにコメントがほとんど付いてないが、これはわざとそうしたのである。けけけけ。

リスト14-2 Z用パレット設定ルーチンとパレット初期化ルーチン

			.Z80	
			.PHASE	0EF00H
1000			;	
1100		BLUE	EQU	1000H
1200		RED	EQU	1100H
		GRN	EQU	1200H
		;		
EF00	DD E5	XPALET:	PUSH	IX
EF02	FE 03		CP	3
EF04	3E 0D		LD	A,13 ;Type mismatch
EF06	C0		RET	NZ
		;		
EF07	78		LD	A,B
EF08	FE 04		CP	4
EF0A	3E 05		LD	A,5 ;Illegal f call
EF0C	D8		RET	C
		;		
EF0D	E1		POP	HL ;DROP
EF0E	01 1FC5		LD	BC,1FC5H
EF11	C5		PUSH	BC
EF12	3E 80		LD	A,80H
EF14	ED 79		OUT	(C),A ;WRITE MODE
		;		
EF16	EB		EX	DE,HL
EF17	7E		LD	A,(HL)
EF18	23		INC	HL
		;(HL)=?.G,A=R.B		
EF19	ED 67		RRD	;A=R.G,(HL)=B.?
EF1B	0F		RRCA	
EF1C	0F		RRCA	
EF1D	0F		RRCA	
EF1E	0F		RRCA	
EF1F	4F		LD	C,A ;A=G.R
EF20	7E		LD	A,(HL) ;C=G.G!!
EF21	23		INC	HL
EF22	E6 F0		AND	0F0H
EF24	57		LD	D,A ;D=B.0!!
		;		
		;A=B.0,(HL)=r.b		
EF25	ED 67		RRD	;A=B.b,(HL)=0.r
EF27	06 10		LD	B,HIGH(BLUE) ;=10H
EF29	ED 79		OUT	(C),A
EF2B	04		INC	B ;B=11H
EF2C	7E		LD	A,(HL)
EF2D	23		INC	HL
EF2E	B2		OR	D ;A=B.r
EF2F	ED 79		OUT	(C),A
EF31	04		INC	B
EF32	7E		LD	A,(HL) ;A=?.g
EF33	E6 0F		AND	0FH
EF35	B2		OR	D ;A=B.g
EF36	ED 79		OUT	(C),A
EF38	C1		POP	BC
EF39	AF		XOR	A



EF3A	ED 79		OUT	(C),A	;ACCESS OFF
EF3C	C9		RET		
		;			
EF3D	01 1FC5	XPINIT:	LD	BC,1FC5H	
EF40	C5		PUSH	BC	
EF41	3E 80		LD	A,80H	
EF43	ED 79		OUT	(C),A	
EF45	CD EF4D		CALL	XPMAIN	
EF48	C1		POP	BC	
EF49	AF		XOR	A	
EF4A	ED 79		OUT	(C),A	
EF4C	C9		RET		
		;			
EF4D	11 1010	XPMAIN:	LD	DE,1010H	
EF50	AF		XOR	A	
EF51	67		LD	H,A	
EF52	6F		LD	L,A	
EF53	4F		LD	C,A	
EF54	AF	L1:	XOR	A	
EF55	42	L2:	LD	B,D	;D=10H
EF56	ED 79		OUT	(C),A	
EF58	04		INC	B	
EF59	ED 61		OUT	(C),H	
EF5B	04		INC	B	
EF5C	ED 69		OUT	(C),L	
		;			
EF5E	19		ADD	HL,DE	
EF5F	C6 11		ADD	A,11H	
EF61	30 F2		JR	NC,L2	
		;			
EF63	0C		INC	C	
EF64	C8		RET	Z	
		;			
EF65	AC		XOR	H	
EF66	20 EC		JR	NZ,L1	
EF68	67		LD	H,A	
EF69	2C		INC	L	
EF6A	18 E9		JR	L2	
		;			
			END		

では説明に入る。そこでリスト 14-1 の 580～690 行なわけだ。これがパレット設定の基礎である。

ここでは「PALET C1, C2」に相当することをしているわけである。まず大事なことは I/O の 1FC5<sub>H</sub> のビット 7, ビット 3 である。これは第 0 章を見れば分かるであろうが, turboZ モードでアナログ RGB のパレット設定に先立っては, 「ビット 7 = 1, ビット 3 = 0」としておかなければならないのである。ビット 7 = 1 はアクセス ON, ビット 3 = 0 は書き込みモード (1 なら読み出しモード) ということである。そして, できれば 680 行にもあるように, 設定が済んだ後は「ビット 7 = 0」としてアクセスを OFF にしておくべきである。さもないと, LIST, INIT 命令などの「パレットをいじるが, アナログ RGB には対応していない命令」が, おかしなパレット設定をしてしまうことがある。

その後は, 3 回の OUT 命令を実行するのだが, その方法がちょっと違和感のある形式になっている。が, 一つ例をあげればたちまち分かってしまうだろう。すなわち,

PALET &H0△□, &Hαβγ

は,

OUT &H10△□, &H□γ

OUT &H11△□, &H□β

OUT &H12△□, &H□α



になるのだ。ただし○△□,  $\alpha\beta\gamma$  というのは、それぞれ「1桁の16進数」である。具体的な数字をあげると、

PALET &HABC, &HDEF

は、

OUT &H10AB, &HCF

OUT &H11AB, &HCE

OUT &H12AB, &HCD

となるのだ。「C」が「OUTされる数値」の方に回されているというのが微妙なのである。ま、これだけ言っておけば大丈夫であろう。あとはリスト14-2であるが、これは「RRD」の使い方がなかなかである。この命令は「Aレジスタの下位4ビット」、「(HL)の上位4ビット」、「(HL)の下位4ビット」の三つの間で値を回すという命令である。詳しくは参考文献4を見ていただきたい。で、気を付けなければいけないのが、この命令を実行すると(HL)が書き換わるということである。よって、たとえばリスト14-1の450行を、

X\$=MKIS (C1)+MKIS (C2):A\$=USR0 (X\$)

などすると、X\$の中身が書き換わってしまうのである。というわけで、この点は注意するよーに。

次にリスト14-3である(100~200行と480行以降はリスト14-1と同じである)。これは64色×2画面モードである。280~370行で64色モード用のパレットを設定し、390~470行で二つのグラフィック画面+キャラクタのプライオリティ(優先順位)をパコパコと切り換えている(ついでにテキストパレットもランダムに変えている)。何かキーを押すと終了する。そいでもって、このときのパレットの設定であるが、「R, G, Bそれぞれの下位2ビットが無効」ということになっている。すなわち、

青→0001 0000 \*\*×× \*\*×× : \*\*×× \*\*\*\*

赤→0001 0001 \*\*×× \*\*×× : \*\*×× \*\*\*\*

緑→0001 0010 \*\*×× \*\*×× : \*\*×× \*\*\*\*

10<sub>H</sub>, 11<sub>H</sub>, 12<sub>H</sub> G R B G', R', B'

において、「\*」が有効なビットで、「×」が無効なビットということである。「×」の部分は注意深く避けないと、予期せぬパレットが設定されたりする。なぜかという、たとえば「OUT &H10FF, &HF?」と「OUT &H10CC, &HC?」は同じだからなのだ(無効ビットを0として考えれば分かる)。よって、前者と後者の両方を実行して、なおかつ「?」の部分が双方で違っていたなら、「あれ? パレットがヘンだぞ」ということになるのである。ま、ここらへんは結構複雑で面倒臭いから、しばらくあれこれといじってみるのが一番手っ取り早いであろう。

#### リスト14-3 64色×2 サンプル

```
100 CLEAR &HEF00
110 MEMS(&HEF00,16)=HEXCHR$("DD E5 FE 03 3E 0D C0 78 FE 04 3E 05 D8 E1 01 C5")
120 MEMS(&HEF10,16)=HEXCHR$("1F C5 3E 80 ED 79 EB 7E 23 ED 67 0F 0F 0F 0F 4F")
```



```

130 MEM$(&HEF20,16)=HEXCHR$("7E 23 E6 F0 57 ED 67 06 10 ED 79 04 7E 23 B2 ED")
140 MEM$(&HEF30,16)=HEXCHR$("79 04 7E E6 0F B2 ED 79 C1 AF ED 79 C9 01 C5 1F")
150 MEM$(&HEF40,16)=HEXCHR$("C5 3E 80 ED 79 CD 4D EF C1 AF ED 79 C9 11 10 10")
160 MEM$(&HEF50,16)=HEXCHR$("AF 67 6F 4F AF 42 ED 79 04 ED 61 04 ED 69 19 C6")
170 MEM$(&HEF60,16)=HEXCHR$("11 30 F2 0C C8 AC 20 EC 67 2C 18 E9 00 00 00 00")
180 DEFUSR0=&HEF00
190 '
200 INIT:KLIST 0
210 OPTION SCREEN 0:WIDTH 40,25,0,1
220 OUT &H1FB0,&H80
230 GOSUB"CLS12"
240 CALL &HEF3D 'XPINIT
250 SCREEN,0:SYMBOL (0,0),"0",8,4,1,0
260 SCREEN,2:SYMBOL (30,30),"2",8,4,2,0
270 '
280 OUT &H1FB0,&H90
290 FOR G=0 TO 15 STEP 4
300   FOR R=0 TO 15 STEP 4
310     FOR B=0 TO 15 STEP 4
320       C1=G*&H100+R*&H10+B
330       C2=C1
340       A$=USR0(MKI$(C1)+MKI$(C2))
350     NEXT
360   NEXT
370 NEXT
380 '
390 SCREEN 0,0:LOCATE 0,5:PRINT STRING$(12,"@")
400 FOR J=0 TO 8 STEP 8
410   FOR I=0 TO 2
420     D=16+J+I:LOCATE 20,15:PRINT HEX$(D)
430     OUT &H1FC0,D:PAUSE 15
440   NEXT
450 NEXT
460 IF INKEY$="" THEN 400
470 END
480 '
490 LABEL"CLS12"
500 SCREEN,0:CLS4
510 SCREEN,1:CLS4
520 SCREEN,2:CLS4
530 SCREEN,3:CLS4
540 INIT
550 RETURN

```

turboZ では上記のプログラムで示した以外にもさまざまな画面モードがあるわけだ。それらの場合にはどんな色数が使えて、パレットの設定はどうするかという問題がある。それにいちいち答えるのも面倒なので、一番おいしそうな「640×400 で 4096 色中から 8 色を使用可能」をリスト 14-4 に示す。

リスト 14-4 640×400 で 4096 色中 8 色モード

```

100 CLEAR &HEF00
110 MEM$(&HEF00,16)=HEXCHR$("DD E5 FE 03 3E 0D C0 78 FE 04 3E 05 D8 E1 01 C5")
120 MEM$(&HEF10,16)=HEXCHR$("1F C5 3E 80 ED 79 EB 7E 23 ED 67 0F 0F 0F 0F 4F")
130 MEM$(&HEF20,16)=HEXCHR$("7E 23 E6 F0 57 ED 67 06 10 ED 79 04 7E 23 B2 ED")
140 MEM$(&HEF30,16)=HEXCHR$("79 04 7E E6 0F B2 ED 79 C1 AF ED 79 C9 01 C5 1F")
150 MEM$(&HEF40,16)=HEXCHR$("C5 3E 80 ED 79 CD 4D EF C1 AF ED 79 C9 11 10 10")
160 MEM$(&HEF50,16)=HEXCHR$("AF 67 6F 4F AF 42 ED 79 04 ED 61 04 ED 69 19 C6")
170 MEM$(&HEF60,16)=HEXCHR$("11 30 F2 0C C8 AC 20 EC 67 2C 18 E9 00 00 00 00")
180 DEFUSR0=&HEF00
190 '
200 INIT:KLIST 0
210 OUT &H1FB0,&H80
220 OPTION SCREEN 0:WIDTH 80,25,1,2
230 CLS4
240 '
250 FOR I=0 TO 7
260   C1=((I ¥ 4) AND 1)*&H800+((I ¥ 2) AND 1)*&H80 +(I AND 1)*&H8
270   C2=((I ¥ 4) AND 1)*&HF00+((I ¥ 2) AND 1)*&HF0 +(I AND 1)*&HF
280   A$=USR0(MKI$(C1)+MKI$(C2))
290 NEXT
300 '
310 FOR I=0 TO 7
320   LINE(I*30,I*20)-(100+I*30,100+I*20),PSET,I,BF
330 NEXT

```



```

340 '
350 FOR I=0 TO 7
360 C1=((I ¥ 4) AND 1)*&H800+((I ¥ 2) AND 1)*&H80 +(I AND 1)*&H8
370 C2=RND(1)*4096
380 A$=USR0(MKI$(C1)+MKI$(C2))
390 NEXT
400 GOTO 350

```

で、これらの基本は、

- 1) 低解像度 (15kHz) のときは、多色モードにするとバンク 0, バンク 1 が重ね合わさって一つの色になる
- 2) 高解像度 (24kHz) のときは、多色モードにするとバンク 0, バンク 1 が別々になり、同時に両方の面を見ることができない。それに伴い、1FC0<sub>H</sub>番地の優先順位などは意味を持たなくなる (キャラクタとの間の優先順位はあるが)
- 3) 色数は、320×200 の 4096 色モード以外は、640×400 のときが 8 色なだけで後はすべて 1 画面 64 色である
- 4) パレットの指定は常に RGB の下位ビットから無効になっていく

なお補足しておくが、Z では IN 命令により、その時点でのパレット設定値の読み出しも可能になっている。X1/turbo では値を読み出せなかっただけではなく、パレットの設定が変わってしまったりしていたのである。これは特に大きなメリットがあるわけでもないが (もちろん CZ-8FB03 の PSAVE などおいしいが)、とりあえずは前進であろう。具体的な、パレット設定値の読み出し方法は、

- 1) 1FC5<sub>H</sub> の第 7 ビットと第 4 ビットを 1 にする。これで多色モードかつ、パレットは読み出しモードとなる
- 2) OUT &H10GR, B? を実行する。「?」は何でもよい。このとき、パレットは読み出しモードなので、新しい値の設定はされないことに注意。この後、B'=INP (&H10GR) AND &H0F とする
- 3) 「&H10」を「&H11」に置き換えて R' を得る
- 4) 「&H10」を「&H12」に置き換えて G' を得る
- 5) こうして得た &HG' R' B' がパレットの設定値

ということになる。

ところで、テキストパレットはそれぞれに 64 色 (階調) を指定できるのであるが、美しくないことにカラー 0 = 黒は黒のまんまなのである。I/O アドレスから見ると 1FB8<sub>H</sub> 番地が臭いのであるが、残念ながらだめである。すなわち、この機能を使ってキャラクタの黒色を他の色に変えることはできないよーである。

で、極めて当然のことながら、テキストパレットは PCG にもかかわってくるのである。よって、turboZ では、PCG で使える色数が大盤振舞されているのである。

## 画像取り込み

これはなかなか画期的な機能であるが、使い方はものすごく簡単なのでプログラムだけ載せておく。まずリスト 14-5 は単純に画像取り込みをしつつ、位置補正をするプログラム



である。110 行の所でマイナスの値を入力すると取り込みをやめるのである。ちなみに、CRT3 というのはスーパーインポーズをする命令である。turboZ で画像取り込みをするためには、スーパーインポーズ画面でなくてはいけないのだ。

リスト 14-5 画像取り込み+位置補正

```
100 WIDTH40,25,0,1:OUT&H1FB0,&H88:CRT 3
110 INPUT A
120 IF A>=0 THEN OUT &H1FC1,A:GOTO 110
130 OUT &H1FB0,&H80:CRT 1
140 END
```

次にリスト 14-6 である。これはモザイク処理である。入力は 8 進数で行なうことと、X、Y 方向のモザイク指定には無効な部分があることに注意。ちなみに「56」を入力して大きなモザイクにした後で **[SHIFT]** + **[.]** でコンピュータ画面にすると、それはそれなりに楽しい画面となる。入力の際に 8 進数ではなく「..」を入力するとクロマキーのデモとなる。これはさりとて面白いわけではない。無限ループだから適当な所で BREAK すること。

リスト 14-6 モザイク+クロマキー サンプル

```
100 WIDTH40,25,0,1:OUT&H1FB0,&H88:CRT 3
110 OUT &H1FC2,0
120 OUT &H1FC3,0
130 OUT &H1FC1,40
140 INPUT A$:IF A$=".." THEN 180
150 A=VAL("&O"+A$)
160 OUT &H1FC2,A:GOTO 140
170 '
180 OUT &H1FC2,0
190 OUT&H1FB0,&H80:CRT 3
200 FOR X=&H80 TO &HC0 STEP &H40
210   FOR I=0 TO 7
220     C=(I AND 4)*8+(I AND 2)*4+(I AND 1)*2
230   PRINTBIN$(X OR C)
240   BEEP:OUT &H1FC3,X OR C:PAUSE 10
250   NEXT
260 NEXT
270 GOTO 200
```

そして最後がリスト 14-7 のスクロールである。turboZ にはスクロール IN/OUT という機能が付いているが、そのサンプルである。お勧めの入力は 8, 9, 10, 11, 12 などである。そのうち「12」は、CRT 出力をカットすることになる。これは別にスクロールと組み合わせる必要はなく、画像取り込み中であろうが、スーパーインポーズ中であろうが、普通のコンピュータ画面であろうが、とにかく CRT 出力をカットしてしまう。ビデオマニアにはおいしいであろう。ところでなかなか面白いのがスクロールとモザイク処理付きの画像取り込みの組み合わせである。モザイク処理部分がクルクルと回ってなかなか面白い。

以上であるが、細かい点であちこちに取りこぼしもあるだろう。一つには turboZ の拡張機能は、カスタム LSI を使ったものであるために、面白そうなもの、便利そうなものを何



リスト 14-7 スクロール サンプル

```
100 WIDTH40,25,0,1:CRT 3
110 SCROLL 2
120 INPUT N:IF N<0 THEN 160
130 OUT &H1FC4,N
140 GOTO 120
150 '
160 SCROLL 0
170 OUT &H1FC4,0
```

でもかんでも寄せ集めたという感じだからでもある。たとえば画像取り込みに、「反転階調」などというものも指定できるのであるが、どう考えてもこれの用途は、反転をボカシの代わりにしたその筋のビデオを、もう一度その筋する機能ぐらいの意味しかないように思うのだが、どうであろうか（今度やってみよ）。それ以外にもやり残したのに「インタレーススーパーインポーズ」もあるが、これはビデオ編集専用であろう。その方面の趣味を持っている方はスクロール IN/OUT の機能とともに大変おいしいであろう。てなところで turboZ であった。



# 付 録

A ダンプリストチェック用プログラム

B X1処理技術者試験

C 初出一覧・参考文献

項目	内容	備考
1. 試験科目	ダンプリストチェック用プログラム	
2. 試験時間	1時間	
3. 試験場所	試験会場	
4. 試験方法	筆記	
5. 試験内容	ダンプリストチェック用プログラムの作成	
6. 試験結果	合格	
7. 試験費用	試験料	
8. 試験結果の通知	試験結果通知書	
9. 試験結果の活用	試験結果の活用	
10. 試験結果の公表	試験結果の公表	
11. 試験結果の活用	試験結果の活用	
12. 試験結果の公表	試験結果の公表	
13. 試験結果の活用	試験結果の活用	
14. 試験結果の公表	試験結果の公表	
15. 試験結果の活用	試験結果の活用	
16. 試験結果の公表	試験結果の公表	
17. 試験結果の活用	試験結果の活用	
18. 試験結果の公表	試験結果の公表	
19. 試験結果の活用	試験結果の活用	
20. 試験結果の公表	試験結果の公表	
21. 試験結果の活用	試験結果の活用	
22. 試験結果の公表	試験結果の公表	
23. 試験結果の活用	試験結果の活用	
24. 試験結果の公表	試験結果の公表	
25. 試験結果の活用	試験結果の活用	
26. 試験結果の公表	試験結果の公表	
27. 試験結果の活用	試験結果の活用	
28. 試験結果の公表	試験結果の公表	
29. 試験結果の活用	試験結果の活用	
30. 試験結果の公表	試験結果の公表	
31. 試験結果の活用	試験結果の活用	
32. 試験結果の公表	試験結果の公表	
33. 試験結果の活用	試験結果の活用	
34. 試験結果の公表	試験結果の公表	
35. 試験結果の活用	試験結果の活用	
36. 試験結果の公表	試験結果の公表	
37. 試験結果の活用	試験結果の活用	
38. 試験結果の公表	試験結果の公表	
39. 試験結果の活用	試験結果の活用	
40. 試験結果の公表	試験結果の公表	
41. 試験結果の活用	試験結果の活用	
42. 試験結果の公表	試験結果の公表	
43. 試験結果の活用	試験結果の活用	
44. 試験結果の公表	試験結果の公表	
45. 試験結果の活用	試験結果の活用	
46. 試験結果の公表	試験結果の公表	
47. 試験結果の活用	試験結果の活用	
48. 試験結果の公表	試験結果の公表	
49. 試験結果の活用	試験結果の活用	
50. 試験結果の公表	試験結果の公表	
51. 試験結果の活用	試験結果の活用	
52. 試験結果の公表	試験結果の公表	
53. 試験結果の活用	試験結果の活用	
54. 試験結果の公表	試験結果の公表	
55. 試験結果の活用	試験結果の活用	
56. 試験結果の公表	試験結果の公表	
57. 試験結果の活用	試験結果の活用	
58. 試験結果の公表	試験結果の公表	
59. 試験結果の活用	試験結果の活用	
60. 試験結果の公表	試験結果の公表	
61. 試験結果の活用	試験結果の活用	
62. 試験結果の公表	試験結果の公表	
63. 試験結果の活用	試験結果の活用	
64. 試験結果の公表	試験結果の公表	
65. 試験結果の活用	試験結果の活用	
66. 試験結果の公表	試験結果の公表	
67. 試験結果の活用	試験結果の活用	
68. 試験結果の公表	試験結果の公表	
69. 試験結果の活用	試験結果の活用	
70. 試験結果の公表	試験結果の公表	
71. 試験結果の活用	試験結果の活用	
72. 試験結果の公表	試験結果の公表	
73. 試験結果の活用	試験結果の活用	
74. 試験結果の公表	試験結果の公表	
75. 試験結果の活用	試験結果の活用	
76. 試験結果の公表	試験結果の公表	
77. 試験結果の活用	試験結果の活用	
78. 試験結果の公表	試験結果の公表	
79. 試験結果の活用	試験結果の活用	
80. 試験結果の公表	試験結果の公表	
81. 試験結果の活用	試験結果の活用	
82. 試験結果の公表	試験結果の公表	
83. 試験結果の活用	試験結果の活用	
84. 試験結果の公表	試験結果の公表	
85. 試験結果の活用	試験結果の活用	
86. 試験結果の公表	試験結果の公表	
87. 試験結果の活用	試験結果の活用	
88. 試験結果の公表	試験結果の公表	
89. 試験結果の活用	試験結果の活用	
90. 試験結果の公表	試験結果の公表	
91. 試験結果の活用	試験結果の活用	
92. 試験結果の公表	試験結果の公表	
93. 試験結果の活用	試験結果の活用	
94. 試験結果の公表	試験結果の公表	
95. 試験結果の活用	試験結果の活用	
96. 試験結果の公表	試験結果の公表	
97. 試験結果の活用	試験結果の活用	
98. 試験結果の公表	試験結果の公表	
99. 試験結果の活用	試験結果の活用	
100. 試験結果の公表	試験結果の公表	



## 付録A ダンプリストチェック用プログラム

リストA-1 はダンプリストチェック用のプログラムである。ごちゃごちゃしているので、打ち込みには十分に注意すること。リストA-2が機械語部分のアセンブルリスト、リストA-3 がその部分を自分自身で表示させたものである。

リストA-1 ダンプリストチェックツール

```

100 T=&HC000: CLEAR T
110 MEM$(T+0,16)=HEXCHR$("05 C8 EB E5 56 23 5E 23 05 28 1C EB 0E 80 1A A1")
120 MEM$(T+16,16)=HEXCHR$("28 01 37 ED 6A 30 08 3E 10 AC 67 3E 21 AD 6F CB")
130 MEM$(T+32,12)=HEXCHR$("09 30 EB 13 10 E8 EB E1 73 23 72 C9")
140 DEFUSR0=T
150 DEFINT B-Z: DIM X(7)
160 INPUT A$
170 B=VAL("&H"+MID$(A$,2,4)): IF B>=0 THEN 160
180 C=VAL("&H"+MID$(A$,7,4)): IF C>=0 THEN 160
190 FOR D=B TO C STEP 128
200 IF D+127<C THEN E=127 ELSE E=C-D
210 Z$=LEFT$(MEM$(D,E+1)+STRING$(127,0),128)
220 F=0: FOR J=0 TO 7: X(J)=0: NEXT
230 FOR I=0 TO 15
240 IF D+F<=C THEN E$=HEX$(D+F)+" ": GOSUB 360 ELSE I=99: GOTO 290
250 Y=0: FOR J=0 TO 7
260 G=ASC(MID$(Z$,F+1,1)): Y=Y+G: X(J)=X(J)+G
270 IF F<=E THEN E$=RIGHT$("0"+HEX$(G),2)+" " ELSE I=99: E$=SPACE$(3)
280 GOSUB 360: F=F+1: NEXT: E$="": "+RIGHT$("0"+HEX$(Y),2): GOSUB 340
290 NEXT: E$=STRING$(33,"-"): GOSUB 340: E$="SUM: ": GOSUB 360
300 FOR J=0 TO 7: E$=RIGHT$("0"+HEX$(X(J)),2)+" ": GOSUB 360: NEXT
310 Z$=USR0(LEFT$(Z$,E+1)): IF E=0 THEN Z$=Z$+Z$
320 E$=RIGHT$("000"+HEX$(CVI(LEFT$(Z$,2))),4): GOSUB 340: E$="": GOSUB 340
330 NEXT: END
340 GOSUB 360: IF LEFT$(A$,1)="P" THEN LPRINT ELSE PRINT
350 RETURN
360 IF LEFT$(A$,1)="P" THEN LPRINT E$: ELSE PRINT E$:
370 RETURN

```

リストA-2 CRC 計算プログラム

			.280	
			.PHASE	0C000H ;OR ANY PLACE
			;	
			;	DE=TOP ADDR
			;	B=LENGTH
C000	05		MKCRCS:	DEC B
C001	C8			RET Z ;LEN=1 THEN RET
			;	
C002	EB		EX	DE,HL
C003	E5		PUSH	HL ;SAVE VAL ADDR.
C004	56		LD	D,(HL)
C005	23		INC	HL
C006	5E		LD	E,(HL)
C007	23		INC	HL
			;	
C008	05		DEC	B
C009	28 1C		JR	Z,FINIS ;JUST 2
			;	
C00B	EB		EX	DE,HL ;HL=SEED,DE=ADDR.
C00C	0E 80		LD	C,80H ;MASK
			;	
C00E	1A		LOOP:	LD A,(DE) ;DATA
C00F	A1			AND C ;CHECK
C010	28 01		JR	Z,SKIP
			;	
C012	37		SKIP:	SCF ;CY=1
C013	ED 6A			ADC HL,HL ;HL=HL+HL+CY
C015	30 08		JR	NC,NEXT ;NEXT BIT



		;			
C017	3E 10		LD	A,10H	
C019	AC		XOR	H	
C01A	67		LD	H,A	
C01B	3E 21		LD	A,21H	
C01D	AD		XOR	L	
C01E	6F		LD	L,A	;HL=HL XOR 1021H
		;			
C01F	CB 09	NEXT:	RRC	C	;ROTATE MASK
C021	30 EB		JR	NC,LOOP	;RIGHT BIT
C023	13		INC	DE	;NEXT BYTE
C024	10 E8		DJNZ	LOOP	;DEC COUNT
		;			
C026	EB		EX	DE,HL	
C027	E1	FINIS:	POP	HL	;GET ADDR.
C028	73		LD	(HL),E	
C029	23		INC	HL	
C02A	72		LD	(HL),D	;SAVE
		;			
C02B	C9		RET		
		;			
			END		

リスト A-3 出力サンプル

C000	05	C8	EB	E5	56	23	5E	23	:	97
C008	05	28	1C	EB	0E	80	1A	A1	:	7D
C010	28	01	37	ED	6A	30	08	3E	:	2D
C018	10	AC	67	3E	21	AD	6F	CB	:	69
C020	09	30	EB	13	10	E8	EB	E1	:	FB
C028	73	23	72	C9					:	D1
-----										
SUM:	BE	F0	02	D7	FF	68	DA	AE	C076	

使い方を説明する。まずはリスト A-1 を間違いないように打ち込む。特に 110 行～130 行は注意して打ち込むこと。打ち込み終わったら **RUN**する前に**セーブ**しておく。そして、そのディスク（もしくはテープ）を取り出しておく。その後に RUN して動作チェックを行なう。RUN した後、「DC000 C02B」でリスト A-3 のように表示したらひとまずは OK である（あくまでひとまずである。まだまだバグが入っている可能性がある）。打ち込み間違いがないということはまずあり得ないから、一発で動くとは期待しないように。バグを取っている間、暴走させてディスクを壊したりしないように、くれぐれも注意していただきたい。

で、プリンタに打ち出す場合は「PC000 C02B」のように「P 開始アドレス 終了アドレス」である。アドレスは常に 4 桁の 16 進数で指定すること。またこのプログラムでは 0000～7FFF のアドレスは指定できないようになっている（必要もない）。

チェックするポイントは、リスト A-3 で になっている部分である。特に右下隅の 4 桁の 16 進数は大事である。

また、このプログラムで第 10 章の MML プログラムをチェックするときは、前もって NEW ON &HD000 を実行しておいてからリスト A-1 と MML プログラムをロードし、100 行の「CLEAR T」を削除してから RUN していただきたい。

余談であるが、100 行の T = &HC000 は、別に C000 でなくてもよく、D000、E000、F000 などでもかまわない。よーするに**チェックする機械語プログラムと重ならないアドレスならどこでもよい**のである。



## 付録B X1 処理技術者試験

最近は軟弱なパソコンユーザーが増えている。

そんなことではいけないので、ここにとりあえずX1 処理試験を用意することにした。おのこの段階に応じて、第2種その筋、第1種その筋、特種その筋が認定される。ただし面倒臭いので、勝手に採点して勝手に認定されたい。

なお、試験問題にはロクでもないものも含まれているので、心してかかるように。

### 注意事項

試験時間は無制限の金網デスマッチ 3 本勝負である。リターンマッチも可であるから、大いに頑張っていたきたい。ただし、ずるはしないように。

### 1. 適性試験

問 以下の項目に YES/NO で答えよ

- (1) 必要ないと分かっているけど、ついつい最適化してしまう
- (2) どんなプログラムでも「もう 1 バイト短くできる」と信じている
- (3) どんなプログラムでも「もう 1 クロック速くできる」と信じている
- (4) 指を 10 ビットのメモリとして使ったことがある
- (5) プログラミングしながら日の出を迎えることは、それほど珍しくない
- (6) CRT は目に悪いと体感している
- (7) 戦闘機のフライトシミュレータは面白そうなゲームである
- (8) 10 進法は不自然である
- (9) 頭の中に裏レジスタがある
- (10) 「CD」とは音楽用語ではなく CALL のことである
- (11) 「地震 雷 火事 停電」である
- (12) 頭の中にフラグレジスタがある
- (13) ディスケットの磁性面に触るようなやつは死刑にすべきだ
- (14) パソコンの前で食事することが多い
- (15) プリンタ用紙にメモすることが多い
- (16) BASIC よりも機械語の方が便利だ
- (17) 老後の趣味はプログラミングと決めている

以上 17 項目のうち 9 項目以上に YES と答えた方はその筋陽性ですので、第 2 種その筋を名乗ることを許可します。

### 2. 第 1 種その筋試験

問 1 PSG のレジスタ設定について答えよ

- a) 花子さんはおかあさんから X1 の PSG を使って 300Hz の音を出して欲しいとたのまれました。チャンネル A を使うとしたら、 $R_0$ ,  $R_1$  にはどんな数を設定すれば、一



番 300Hz に近い周波数の音が出るでしょうか。

b) 太郎君はその筋なので、X1 を使って出せる一番低い音（周波数の小さい音）を出してみようと思いました。さて、太郎君が出した音の周波数はいくつでしょう。小数点第 2 位で四捨五入して答えなさい。

## 問 2 パレットの設定について答えよ

NEW BASIC と turbo BASIC のコマンド「PALET@」は一つのコマンドで最大八つのパレットを設定できるコマンドである(知らなかったら×)。

そこで「PALET@文に相当する 3 バイトのデータ」について答えよ。

- a) 資料を見ずに、初期状態の「3 バイト」を答えよ。また、その 3 バイトはそれぞれ何番地に OUT するか？
- b) 「PALET@ 4, 1, 2, 6, 4, 1, 2, 6」に対応する 3 バイトは？
- c) グラフィックを 4 色だけに制限する代わりに、青>赤>緑(+黒)の優先順位で使えるようにしたい(勝手な話だが、この時点で何を言おうとしているのか分からなかったら×)。青を PLANE 0(4000<sub>H</sub>~7FFF<sub>H</sub>)、赤を PLANE 1(8000<sub>H</sub>~BFFF<sub>H</sub>)、緑を PLANE 2(C000<sub>H</sub>~FFFF<sub>H</sub>)とするには、どうすればよいか。「PALET@ ?, ?, ?, ?, ?, ?, ?, ?」と「3 バイトのデータ」で答えよ。
- d) c) と似たケースで、白(PLANE0)>青(PLANE1)>赤(PLANE2) (+黒) の場合について答えよ。
- e) 4 色ではなく 5 色に制限して青、赤、緑>白(+黒)としたい。パレットコードを青=1, 赤=2, 緑=3>白=4 (黒=0) とするにはどうするか。「PALET@ ?, ?, ?, ?, ?, ?, ?, ?」と「3 バイトのデータ」で答えよ。

## 問 3 次の式をウンウン言いながら暗算して16進数で答えよ。ただし~<sub>B</sub> は 2 進数, ~<sub>O</sub> は 8 進数を表すものである。

- a)  $FA_H + 27_H + 34_H$
- b)  $FF_H \times 1011_B$
- c)  $774_O + 444_O$
- d)  $10_H \times 11_H + 2A_H$
- e)  $0F_H \times 0F_H$
- f)  $716_H \div 2$
- g)  $\sqrt{100_H}$
- h)  $\sqrt{144_H}$
- i)  $\sqrt{2710_H}$

## 問 4 次のサブルーチンの実行時間を計算し、クロック数(ステート数)と時間( $\mu$ sの単位)を答えよ。もちろんクロックは 4MHz とする。ただし Z80 の解説書を参照してもよい。



a)

```
LD      A, 100
LOOP: DEC A
RET     Z
JR      LOOP
```

b)

```
LD      A, 100
LOOP: DEC A
JP      NZ, LOOP
RET
```

c)

```
LD      A, 100
LOOP: DEC A
JR      NZ, LOOP
RET
```

d)

```
PUSH    AF
PUSH    DE
LD      DE, 151
LOOP: DEC DE
LD      A, D
OR      E
JR      NZ, LOOP
JR      SKIP
SKIP: OR A
POP     DE
POP     AF
RET
```

問5 HLレジスタ×msだけ(よーするに HL=1なら1ms, HL=2なら2ms) 時間つぶしをするサブルーチンを作りたい (HL=0のときは考えなくてよい)。ただし、呼び出す側での、

```
PUSH    HL
LD      HL,    nm    ; 何m秒か
CALL    そのサブルーチン
```



POP HL

の時間も入れて考えるとする。

a) 空いてる所をその筋な文字列で埋めなさい (甘えは許しません)

呼び出し例			
	PUSH	HL	;11
	LD	HL, 1234	;10
	CALL	WAIT	;17
	POP	HL	;10
サブルーチン			
WAIT:	PUSH	AF	;11
	PUSH	DE	;11
	NEG		;8
	DEC	HL	;6
	LD	A, H	;4
	OR	L	;4
	;		
	JR	(1), CASE0	;12/7
	LD	DE, (2)	;10
	CALL	WS	;17
	BIT	0, (HL)	;12
	DEC	DE	;6
	OR	A	;4
	;		
LOOP:	LD	DE, (3)	;10
	NEG		;8
	NEG		;8
	CALL	WS	;17
	DEC	(4)	;6
	LD	(5)	;4
	OR	(6)	;4
	JR	NZ, LOOP	;12/7
	;		
FINIS:	POP	DE	;10
	POP	AF	;10
	RET		;10
	;		
CASE0:	LD	DE, (7)	;10
	CALL	WS	;17
	JR	FINIS	;12
	;		
WS:	DEC	DE	;6
	LD	A, D	;4
	OR	L	;4
	JR	NZ, WS	;12/7
	RET		;10

b) 上記のプログラムを短く作り変えなさい (これは簡単)

### 3. 特種その筋試験 (論文)

a) 第14章のリスト14-2 (turboZの4096色パレットの初期化)のEF4D<sub>H</sub> ~ EF6B<sub>H</sub>の部分の動作を理解し解説せよ。

b) それと同機能で、より短いサブルーチンを作れ。もしも不可能と考える場合は、不可能であることを証明せよ。



**【解答】**

**第1種その筋試験**

**問1**

- a)  $R_0 = 161, R_1 = 1$
- b) 30.5Hz

**問2**

- a)  $AA_H, CC_H, F0_H$  の3バイト  
OUTする番地は,  
 $AA_H \rightarrow 1000_H$  番地  
 $CC_H \rightarrow 1100_H$  番地  
 $F0_H \rightarrow 1200_H$  番地
- b)  $22_H, CC_H, 99_H$
- c) PALET@ 0, 1, 2, 1, 4, 1, 2, 1  
 $AA_H, 44_H, 10_H$
- d) PALET@ 0, 7, 1, 7, 2, 7, 1, 7  
 $EE_H, BA_H, AA_H$
- e) PALET@ 0, 1, 2, 4, 7, 1, 2, 4  
 $32_H, 54_H, 98_H$

**問3**

- a)  $155_H$     b)  $AF5_H$     c)  $320_H$     d)  $13A_H$     e)  $E1_H$     f)  $38B_H$     g)  $10_H$
- h)  $12_H$     i)  $64_H$

**問4**

- a) 2695クロック,  $673.75\mu s$
- b) 1417クロック,  $354.25\mu s$
- c) 1612クロック,  $403\mu s$
- d) 3999クロック,  $999.75\mu s$  (けけけけけ)

**問5**

- a)
- (1) Z    (2) 147    (3) 151    (4) HL    (5) A, H    (6) L    (7) 147
- b)

自己採点しなさい。②と⑦がともに「147」であることを使えば簡単。もちろんそれ以外でもよい。

**特種その筋試験**

解答が載っているなどと思っていた人は失格。



## 付録C 初出一覧・参考文献

### 【初出一覧】

各章の Oh!MZ 誌連載時における掲号とタイトルを下記に示す。

- 第0章 きっと完全無欠な I/O マップ——'85年6月号「第1回 たぶん完全無欠な I/O マップ」、'87年2月号「第21回 ほとんど完全無欠な I/O マップ」
- 第1章 CRTC でどすこいである——'85年7月号「第2回 そこに CRTC があるからなのだ」
- 第2章 PCG は二度おいしいのである——'86年3月号「第10回 PCG のお通りである」
- 第3章 漢字名野出亜留——'86年11月号「第18回 とーとー漢字なのである」
- 第4章 サブ CUP のおかげなのである——'85年10月号「第5回 サブ CUP は必修科目なのである」
- 第5章 CTC は律儀なのである——'86年4月号「第11回 CTC はきちょーめんなのである」
- 第6章 SIO でマウスである——'86年5月号「第12回 SIO は通信ばかりではないのである」
- 第7章 通信だってするのである——'87年8月号「最終回 通信プログラムである」
- 第8章 DMA はヘビー級である——'85年8月号「第3回 Z80の一族はただ者ではないのである」、'85年9月号「第4回 DMA はグラフィックもしてしまうのである」
- 第9章 ディスクを回すのである——'85年12月号「第7回 軟式円盤の基礎である」、'86年1月号「第8回 FDC は挑戦的である」、'86年2月号「第9回 FDC は業師なのである」、'86年7月号「第14回 DMA にはディスクがよく似合うのである」
- 第10章 PSG は基本である——'85年11月号「第6回 PSG はてりめえである」
- 第11章 FM 音源ナハトムジーク——'87年5月号「第23回 FM 変調するのである」、'87年6月号「第24回 MML を作るのである」、'87年7月号「第25回 MML を完成するのである」
- 第12章 カラーイメージボードで取り込むのである——'87年4月号「第22回 カラーイメージボードなのである」
- 第13章 テープもやってしまうのである——'86年6月号「第13回 あげくのはてにはテープなのである」

### 【参考文献】

1. 「X1 マシン語活用百科」、清水保弘著、産業報知センター刊
2. X1 turbo 回路図公開、「Oh!MZ」、'84年4月号
3. 続インターフェイス LSI の研究 CRTC [HD46505] 編、「マイコンコンピュータ No.8」、千葉憲昭著、CQ 出版社刊
4. 「Z80ファミリ・ハンドブック」、額田忠之著、CQ 出版社刊
5. 「プログラマブルサウンドジェネレータ データマニュアル」、ゼネラル インストルメント インターナショナル コーポレーション (GI) 刊
6. 「YM2151ユーザーズマニュアル」、日本楽器製造株式会社刊
7. 「CZ-8BS1 取扱説明書」、シャープ刊
8. 「X1 システム研究室」、有田隆也、牛嶋昌和、I. Rittaporn 著、日本ソフトバンク刊
9. I/O '85年7月号「マイクロプロセッサを比較する [19]」、Processor's Professor 著、工学社刊
10. 「最新フロッピー・ディスク装置とその応用ノウハウ」、高橋昇司著、CQ 出版社刊
11. 「ガラスの仮面」1～34巻、美内すずえ著、白泉社刊
12. 「小さなお茶会」1～6巻、猫十字社著、白泉社刊







# 索引

## A

ALG .....281  
AM .....208  
AY-3-8910 .....230

## B

BASIC ROM .....13  
BCD .....90  
Bin .....184  
BIOS ROM .....20  
bps(ボー) .....99  
BUSY .....192, 212

## C

CG.....16, 56  
CHDIR.....186  
cluster .....182  
C-MOS .....78  
CP/M .....176  
CRC .....210  
CRTC(CRT コントローラ) .....16, 32, 46  
CR コード .....133  
CTC .....21, 98, 242, 289  
CUDISP 信号 .....36  
cylinder .....180  
CZ-8BM2 .....98, 132  
CZ-8BS1 .....242  
CZ-8BV1 .....284  
CZ-8BV2 .....284  
CZ-8CB01 .....44  
CZ-8FB01 .....44, 196  
CZ-8FB02 .....219, 318  
CZ-8FB03 .....318

CZ-8RL1 .....79, 310  
CZ-8RS .....12

## D

DATA .....210  
DATA REQUEST .....189  
Device dump .....181  
DI .....84  
DISPTMG 信号 .....36  
DMA .....21, 148, 217  
DOS .....193

## E

EMM .....13, 163  
EI .....83

## F

FAT .....182, 184  
FD .....14, 217  
FD1791 ファミリ .....187  
FDC .....187  
FDD .....192  
FM .....179  
FM 音源 .....11, 242  
FM 記録方式 .....179

## G

GAP .....208  
G(グラフィック)-RAM .....28, 163, 226

## H

HD46505-SP.....33  
HLD .....192  
HLT .....192



**I**

IBM フォーマット	176
ID	204, 209
INDEX ホール	205
INP 関数	226
INT	86
I/O	10
IPL	182, 184
IPL ROM	20
IST	245
I レジスタ	86

**J**

JIS 漢字コード	64
-----------	----

**K**

KILL	185
------	-----

**L**

LFO	247
LF コード	133

**M**

M (メガ)	231
MB8876	187
MB8877	187
MFM	179
MFM 記録方式	179
MKDIR	186
MML	242, 260, 276
μPD765 ファミリ	187

**N**

NMI	86
NOT READY	192

**O**

OPM	242
OPN	242

**P**

PCG	16, 25, 41, 44
PIO-3055-01	13
PSG	20, 230

**R**

RAM CG	44
RESTORE	219
ROM CG	44
RS-232C	12, 98, 132

**S**

SCSI	14
sector	181
side	180
SIO	21, 98, 114
Sync	208
Sys	184

**T**

TL	251
track	180
turboZ	11, 318
TV	89
TYPE (FDC コマンド)	188



<b>V</b>	
VIP .....	245
VRAM .....	46, 162

<b>W</b>	
WNBSY .....	191

<b>Y</b>	
YM2203 .....	242

<b>Z</b>	
Z 画像取り込み位置補正指定 .....	23
Z クロマキー指定 .....	24
Z スクロール指定 .....	24
Z テキストパレット指定 .....	22, 324
Z80 .....	99, 114, 148
Z プライオリティ指定 .....	22
Z モード指定 .....	22
Z モザイク取り込み指定 .....	23, 325
Z 量子化取り込み指定 .....	23

<b>あ</b>	
アタックレート .....	246, 251
アトリビュート .....	182
アルゴリズム .....	245

<b>い</b>	
インクリメント .....	220
インターリーブファクタ .....	207
インタラプトレジスタ .....	86
インタレーススーパーインポーズ .....	318
インタレースモード .....	35
インデックスホール .....	178, 205

インフォメーションブロック .....	303
---------------------	-----

<b>う</b>	
ウィンドウ .....	128, 289

<b>え</b>	
エッジ .....	314
エンドコード .....	133
エンド・オブ・ブロック .....	157, 226
エンベロープ .....	230, 245

<b>お</b>	
音階 .....	277
音量 .....	230

<b>か</b>	
階層化ディレクトリ .....	182, 186
解像度 .....	324
外部クロック .....	99
外部 RAM ボード .....	13
外部 ROM .....	13
カウンタ .....	99
カウンタモード .....	101
カセットデッキ .....	90
カナの表現方法指定 .....	133
画面管理 .....	25
カラーイメージボード .....	11, 284
カレンダー .....	90
漢字コード .....	64
漢字 ROM .....	13, 16, 71
漢字 VRAM .....	28, 68

<b>き</b>	
キーデータ .....	88



キー入力 .....84

く

区点コード .....64  
クラスタ(cluster) .....182  
グラフィック .....225  
グラフィック画面 .....68  
グラフィックパレット .....15, 24  
グラフィック(G-)RAM .....28, 163, 166, 226  
クロック .....116  
クロックパルス .....179  
クロックビット .....179  
クロマキー .....318

け

ゲームキー .....84

こ

黒色制御 .....26  
コマンドレジスタ .....192  
コンティニューモード .....156

さ

サーチ .....149, 170  
最初のセクタ .....208  
サイド .....180  
サイド番号 .....182  
サステーンレート .....246  
サステーンレベル .....246  
サブCPU .....17, 78  
サブレジスタ .....153

し

シーク .....192

シークエラー .....195

シークコマンド .....192

システムクロック .....104

シフト JIS 漢字コード .....65

周波数 .....230

ジョイスティック .....20, 230

シリアル通信 .....114

シリアルマウス .....114

シリンダ(cylinder) .....180

シリンダ番号 .....201

す

スーパーインポーズ .....16, 35, 325

スキューファクタ .....207

スクランブル回路 .....284

スクランブルモード .....284

スクロール IN/OUT .....325

スタートビット .....114, 303

スタートポート .....26

ステータス .....189

ステータスレジスタ .....188, 206

ステップ・イン/ステップ・アウト .....200

ステップレート .....194

ストップビット .....114, 133

スムーズスクロール .....35

せ

セクタ(sector) .....181, 205

セクタ番号 .....204

ゼロカウント .....99

そ

ソフトフォーマット .....181, 225



## た

ターミナル	134
タイマ	99
タイマトリガ	99
タイマモード	101
タイマ割り込み	104
タイムアウト	99
タイムコンスタント	99, 116
ダウンカウンタ	100
単密	178

## ち

チェックサムエラー	303
チャンネル	230

## つ

通信	114
通信制御指定	133
通信パラメータ	133

## て

データビット	114, 133
データフィールド	204
テープ	300
テープのリード/ライト	303
ディケイレート	246, 251
ディスクライト	199
ディスクリード	199
ディレクドリ	182
テキストアトリビュート	27
テキストスクロール	166
テキストパレット	318
テキスト VRAM	27, 161

デリーテッドアドレスマーク	203
転送	148, 164

## と

トーン	230
動画面	289
時計	92
ドライブのセレクト	190
トラック	180
トラックレジスタ	188, 190

## に

日本語文字列の表現方法指定	133
---------------	-----

## の

ノイズ	230
-----	-----

## は

80C48	78
80C49	78
バーストモード	155
ハードディスク	13
ハーフトーン回路	284
ハイスピードモノクロ	284
バイトモード	155, 220
倍密	178
バス	148
パスワード	183
パソコン通信	132
パラレルポート	230
パリティ	133
パレット設定	321



## ふ

フィードバック	250
フォースインタラプト	200
フォーマット	181, 208, 218
物理フォーマット	181, 223
プライオリティ	16
プリスケアラ	101
ブロック長	220
フロッピーディスク	176, 210
プロテクト	315

## ほ

ボー (bps)	99
ボーレート	133, 300, 310
ボーレートジェネレータ	21
ポインタビット	154
ボンディングオプション	114

## ま

マーク	302
マウス	114
マウスインターフェイス	98
マウสดライバ	117
マルチセクタ	226
マルチタスク	104, 289
マルチプレクス	220

## み

ミッシングクロック	208
-----------	-----

## め

メイン/バンクメモリ切り換え	12
----------------	----

## も

モーター	190
モードの指定	128
モザイク	325

## よ

4096 色	15, 318
--------	---------

## ら

ライトトラック	200, 207
ライトレジスタ	153
ラスト	34

## り

リーダー	302
リードトラック	200, 207
リードレジスタ	158
リセット	103
立体ボード	11
リリースレート	246, 251

## れ

レコード番号	182, 225
レディ信号	158

## わ

割り込み	84
割り込みベクトル	102



《著者略歴》

E000 24 45 4D 06 24 31 53 AE : 12  
E008 C1 B3 C9 FF 23 FF F0 01 : 4F  
E010 76 FF 79 ED E0 38 01 23 : 17  
E018 E0 2C CD FF E0 38 11 23 : 24  
E020 FF 23 23 FF 00 AB 01 23 : 13  
E028 FF 01 10 01 E0 32 11 09 : 3D  
E030 00 12 01 FF 2A 3E 23 AB : 48  
E038 01 07 A8 21 : D1

-----  
SUM: 3A 60 38 11 11 BB 8A CC D5B4

# 試験に出るX1

ハードウェアのフルコース

定価はカバーに記  
載されております

昭和62年12月15日 初版発行  
昭和63年 6 月15日 第4刷発行

著 者 祝 一平

発行者 孫 正義

発行所 株式会社日本ソフトバンク  
出版事業部

〒102 東京都千代田区九段南2-3-26

☎03(261)4095

印 刷 壮光舎印刷株式会社

©1987 Printed in Japan

ISBN4-930795-88-5

落丁本、乱丁本はお取り替えいたします















パワフル  
**V**  
**turbo**  
SHARP



